UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

NICKSSON CKAYO ARRAIS DE FREITAS

DEEP LEARNING APPROACH FOR TRAJECTORY USER-LINKING IN MULTIDIMENSIONAL AND IMBALANCED DATASETS

FORTALEZA

2024

NICKSSON CKAYO ARRAIS DE FREITAS

DEEP LEARNING APPROACH FOR TRAJECTORY USER-LINKING IN MULTIDIMENSIONAL AND IMBALANCED DATASETS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Ciência de Dados e Inteligência Artificial

Orientador: Prof. Dr. José Antônio Fernandes de Macêdo

Coorientador: Dr. José Florêncio de Queiroz Neto

FORTALEZA

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F937d Freitas, Nicksson Ckayo Arrais de.
Deep Learning Approach for Trajectory User-Linking in Multidimensional and Imbalanced Datasets / Nicksson Ckayo Arrais de Freitas. – 2024.
87 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2024.
Orientação: Prof. Dr. José Antônio Fernandes de Macêdo.
Coorientação: Prof. Dr. José Florêncio de Queiroz Neto.

1. Deep Learning. 2. Machine Learning. 3. Trajectory Classification. 4. Trajectory User-Linking. 5. Imbalanced dataset. I. Título.

CDD 005

NICKSSON CKAYO ARRAIS DE FREITAS

DEEP LEARNING APPROACH FOR TRAJECTORY USER-LINKING IN MULTIDIMENSIONAL AND IMBALANCED DATASETS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Ciência de Dados e Inteligência Artificial

Aprovada em: 19 de agosto de 2024

BANCA EXAMINADORA

Prof. Dr. José Antônio Fernandes de Macêdo (Orientador)
Universidade Federal do Ceará (UFC)

Dr. José Florêncio de Queiroz Neto (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. João Paulo do Vale Madeiro
Universidade Federal do Ceará (UFC)

Prof. Dr. Fábio André Machado Porto
Laboratório Nacional de Computação Científica (LNCC)

Profa. Dra. Livia Almada Cruz
Universidade Federal do Ceará (UFC)

To my beloved wife, Graciella, and my son, Nickolas, who have brought joy and purpose to my life. And to my grandfather, Antônio Noronha de Freitas, who always believed in me and invested in my future, instilling in me the importance of education and the values that guide my life.

This work is dedicated to you with all my heart.

ACKNOWLEDGEMENTS

I praise God for the gift of life, for guiding me through every step of this journey, and for providing me with the strength and perseverance to overcome all challenges along the way.

I express my deepest gratitude to my family, who has been my steadfast support system throughout this journey. To my beloved wife, Graciella Melo de Araújo Freitas, and my dear son, Nickolas Araújo de Freitas, thank you for your endless love, patience, and encouragement. To my parents, Carlos Feitosa de Freitas and Nivea Rivânia Arrais, I am eternally grateful for your unwavering belief in me and the values you instilled in me. To my grandfather, Antônio Noronha de Freitas, and my grandmothers, Maria de Fátima Feitosa and Maria das Neves Arrais, your wisdom and love have guided my life.

I sincerely thank my advisor, Prof. José Antônio Fernandes de Macedo, for your invaluable guidance, patience, and support throughout this research. Your mentorship has been crucial to my development as a scholar. I am also profoundly grateful to my co-advisor, Prof. José Florêncio, for your insightful advice and for always supporting me in my academic endeavors.

I am profoundly thankful to my friends and colleagues who have stood by me and contributed to my research. A special mention goes to Francisco Carlos Freire Nunes—your friendship and support have been invaluable throughout this journey.

I also wish to thank all my colleagues from the Insight Data Science Lab: Regis Pires Magalhães, Ticiane Linhares, Leopoldo Melo, and Lívia Almada —your contributions, whether direct or indirect, have significantly influenced the outcome of this work. I am fortunate to have worked alongside such talented and dedicated individuals.

A special thanks goes to the ARiDa - Advanced Research in Database team. Your innovative spirit, collaboration, and dedication to pushing the boundaries of database research have been a constant source of inspiration. I am grateful for the opportunity to have been part of such an outstanding team.

I would also like to thank all the Department of Computer Science professors at UFC. Your teachings and classes have been fundamental in shaping my academic path and have contributed significantly to my success. I appreciate your dedication to education and inspiring me to pursue excellence in my research.

I would like to thank my thesis committee, Prof. Dr. José Antônio Fernandes de Macedo, Dr. José Florêncio de Queiroz Neto, Dr. Fábio André Machado Porto, Prof. Dr.

João Paulo do Vale Madeiro, and Profa Dra. Livia Almada Cruz for all the considerations that improved this thesis.

To everyone else who has played a role in my academic journey, even if not mentioned by name, know that your support is deeply appreciated.

"Whatever you do, work at it with all your heart, as working for the Lord, not for human masters, since you know that you will receive an inheritance from the Lord as a reward. It is the Lord Christ you are serving."

(Colossians 3:23-24 (NIV))

RESUMO

Neste trabalho, exploramos o problema de classificação de trajetórias, com foco no desafio da Vinculação de Usuários a Subtrajetórias (Trajectory User-Linking - TUL) no contexto de Redes Sociais Baseadas em Localização (Location-Based Social Networking - LBSN). Nosso objetivo é associar subtrajetórias anônimas a usuários específicos em plataformas como o Foursquare, uma tarefa essencial para aprimorar a personalização de serviços, otimizar o planejamento urbano e de negócios e implementar estratégias mais eficazes em saúde pública e segurança. O problema de TUL apresenta diversos desafios: as bases de dados de LBSN geralmente contêm volumes massivos de informações; a representação de dimensões espaciais e temporais em modelos de aprendizado de máquina é complexa; os pontos de trajetória espaço-temporais são esparsos; as trajetórias de LBSN possuem múltiplas dimensões, incorporando características adicionais associadas aos pontos; o número de classes frequentemente excede o número de padrões de movimento, com mais de 100 classes; além disso, os conjuntos de dados podem apresentar distribuições desbalanceadas. Para abordar esses desafios, apresentamos o DeepeST (Deep Learning for Sub-Trajectory Classification), um modelo de aprendizado profundo que utiliza vetores de embeddings inspirados em técnicas de processamento de linguagem natural para lidar com grandes volumes de dados e a esparsidade das subtrajetórias. Até onde sabemos, o DeepeST é o primeiro modelo projetado especificamente para enfrentar os desafios impostos por conjuntos de dados desbalanceados no problema de TUL. Avaliamos o desempenho do DeepeST em três estudos de caso, comparando-o com algoritmos de aprendizado de máquina, como Random Forest e XGBoost, e abordagens de aprendizado profundo do estado da arte, incluindo MARC, BITULER e TULVAE. Os experimentos foram realizados com conjuntos de dados de GPS e LBSN, aplicados a tarefas de vinculação de trajetórias a usuários e classificação de atividades criminais. O DeepeST apresentou resultados significativamente superiores em acurácia balanceada, precisão, revocação (recall) e F1-score em todos os cenários analisados.

Palavras-chave: Vinculação de Usuários a Trajetórias; Classificação de Trajetórias; Aprendizado Profundo; Aprendizado de Máquina; Trajetória esparsa; Conjunto de dados desbalanceado.

ABSTRACT

In this work, we investigate the trajectory classification problem and focus on the Trajectory User-Linking (TUL) challenge within Location-Based Social Networking (LBSN) to associate anonymous subtrajectories with specific users on platforms such as Foursquare. This association is crucial for enhancing service personalization and targeting, optimizing urban and business planning, and facilitating effective public health and safety strategies. The TUL problem presents multiple challenges: LBSN databases often contain large volumes of data; there is complexity in representing spatial and temporal dimensions in machine learning models; spatiotemporal trajectory points are sparse; LBSN trajectories are multidimensional, that is, there are other features available that are associated with trajectory points; the number of classes often exceeds the number of motion patterns (e.g., more than 100); and the datasets may have imbalanced distributions. To address these challenges, we introduce a new deep learning model called DeepeST (Deep Learning for Sub-Trajectory classification), which employs embedding vectors inspired by natural language processing techniques to manage large data volumes and tackle sparsity from subtrajectories. To our knowledge, DeepeST is the first model designed to address the challenges posed by imbalanced datasets in the TUL problem. We evaluated DeepeST's performance through three case studies, comparing it against machine learning algorithms such as Random Forest and XGBoost and state-of-the-art deep learning approaches including MARC, BITULER, and TULVAE, using GPS and LBSN datasets for trajectory-user linking and criminal activities. DeepeST demonstrated significantly higher balanced accuracy, precision, recall, and F1-score values in all experiments.

Keywords: Trajectory User-Linking; Trajectory Classification; Deep Learning; Machine Learning; Sparse trajectory; Imbalanced dataset.

LIST OF FIGURES

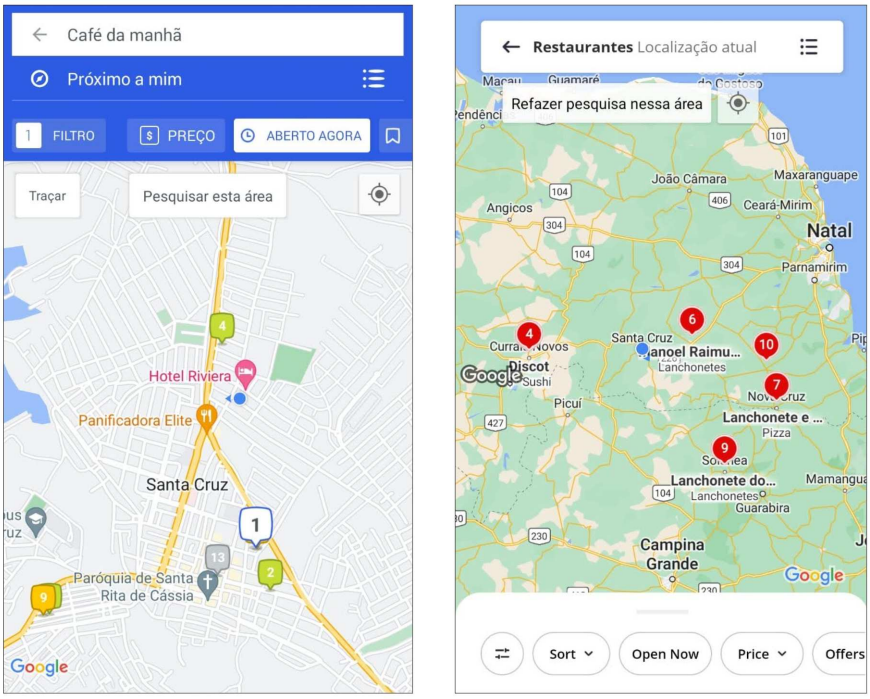
LIST OF TABLES

CONTENTS

1 INTRODUCTION

Social networks are structures composed of individuals interconnected through various forms of interdependence, such as friendships, shared interests, and collective knowledge. Location-Based Social Network (LBSN) platforms stand out by integrating smartphone GPS technology with social networking functionalities. These platforms aim to provide services and experiences tailored to the user's geographical location, including maps, directions, local search, and location-based advertising, as shown in Figure 1. Prominent examples of LBSNs include Foursquare, Yelp, Flickr, ride-hailing apps, food delivery apps, weather apps, and location-sharing features within broader social networks such as Facebook and Instagram.

Figure 1 – Examples of Location-Based Social Networks



Source: Created by the author

Users of LBSN platforms can check in at specific locations, allowing them to share their whereabouts with friends and earn rewards or badges. They can also share reviews and ratings for places they visit, such as restaurants, shops, and attractions, and tag photos with geographic data, creating a visual map of their experiences. Additionally, these platforms provide maps, routes, and navigation assistance to help users find their way to specific locations. Users can search for businesses, services, stops, and food during a trip, thereby creating a rich dataset for movement analysis.

The sequence of user check-ins on an LBSN platform constitutes a trajectory, defined as a set of points organized in chronological order that represents the movement of objects across geographical space (ZHENG, 2015). Analyzing LBSN trajectories offers multifaceted benefits across several sectors. This analysis enhances user experiences through personalized recommendations (WEI; ZHANG, 2020; CHANG *et al.*, 2023; CANTURK *et al.*, 2023), improves urban planning (KHAN *et al.*, 2020), supports public safety (FREITAS *et al.*, 2021c; FREITAS *et al.*, 2021b; GAO *et al.*, 2017), informs transportation systems (YANG, 2023), and drives targeted marketing strategies. Additionally, LBSN data analysis provides economic insights for businesses, supports environmental monitoring (WEI *et al.*, 2022), contributes to social research by examining behavior and cultural trends (RIZWAN *et al.*, 2018), and assists public health officials in tracking disease spread (KIM *et al.*, 2020).

In this work, we investigate the trajectory classification problem because it offers insightful analysis and predictions about the movement patterns (SILVA *et al.*, 2019). The trajectory classification problem involves the challenge of categorizing trajectories—sequences of points representing the movement of an object or entity over time—into predefined classes. These trajectories can represent, for example, the path of a vehicle, the movement of an animal, or a user's navigation on a digital interface. Trajectory classification is used to identify behavioral patterns, predict future movements, or detect anomalies. This problem is addressed in various fields, such as transportation, surveillance, sports, and healthcare, using machine learning techniques and time series analysis to identify and differentiate trajectories based on characteristics like shape, speed, and direction. Basic examples of trajectory classification are:

(i) determining the transportation mode of the moving object like a car, bus, bike, taxi, airplane, or train (ZHENG *et al.*, 2008; BOLBOL *et al.*, 2012; TRAGOPOULOU *et al.*, 2014; VARLAMIS, 2015).

(ii) classify animal categories such as rabbit, dog, and leopard (LEE *et al.*, 2008).

(iii) determines a user's next position like home, school, cafe, office, or restaurant (FREITAS *et al.*, 2021b; FREITAS *et al.*, 2021a).

(iv) identifying the user of a anonymous trajectory (GAO *et al.*, 2017; ZHOU *et al.*, 2018; May Petry *et al.*, 2020).

In this work, we address the Trajectory User Linking (TUL) problem, which involves associating anonymous trajectories with their respective users. The challenge arises when mobility data is anonymized to protect user privacy, such as in Location-Based Social Networks

(LBSNs) and ride-sharing services. The primary motivation to solve the TUL problem involves analyzing human mobility patterns and providing insights into how individuals move and interact with their environment. The analysis encompasses various aspects of human movement, including daily routines, commuting habits, event participation (check-ins), and general movement patterns within specific geographical areas. Successfully linking anonymized trajectories to individual users can enhance the accuracy of location-based recommendations and detect abnormal events from LBSN datasets. From a security perspective, models can identify anomalies in user behavior, such as unusual check-in patterns that may indicate fraudulent activity or potential threats. A proactive approach enables timely interventions without compromising user privacy.

Enhancing personalization is another critical benefit; for instance, applications like Foursquare or Yelp can enable businesses to provide tailored recommendations and promotions based on user movement patterns, even without knowing their identity. This ensures a personalized experience while respecting privacy, increasing customer engagement, and leading to higher sales and profitability. Additionally, optimizing service delivery becomes more feasible as urban planners and service providers can analyze aggregated, anonymized data to improve infrastructure, public transport routes, and resource allocation. These practical applications are crucial for advancing business intelligence and smart city initiatives, demonstrating the importance and relevance of TUL models in providing accurate, secure, and user-centric services across various sectors (CHEN *et al.*, 2021; FENG *et al.*, 2019).

The TUL problem is considered more challenging than other trajectory classification problems for several reasons (GAO *et al.*, 2017):

1. Trajectory databases often contain large volumes of data due to the substantial number of users and extended data collection periods.
2. Representing spatial and temporal data is complex because latitude and longitude coordinates are continuous and highly dimensional variables. Moreover, temporal features are inherently cyclical, such as hours repeating every 24 hours and days cycling annually.
3. The sequence of spatio-temporal points is often intermittent and widely spaced in both time and location (YANG *et al.*, 2015), posing a significant challenge in accurately modeling and predicting user trajectories. This irregularity leads to gaps and inconsistencies in the data, making it challenging to capture continuous movement patterns effectively.
4. Trajectories are multi-dimensional, linked to other properties such as geographic contexts, temporal patterns, behavioral intents, and environmental factors (May Petry *et al.*, 2020).

5. The number of classes can be much larger than the number of motion patterns. For example, the Brightkite dataset contains 51,406 users, and Gowalla possesses 107,092 users (YANG *et al.*, 2015).
6. Trajectory datasets often present imbalanced distributions of the target variable, with an Imbalance Ratio (IR) greater than two (FERNÁNDEZ *et al.*, 2008). For instance, in the TUL problem, the three LBSN datasets (Gowalla, Brightkite, and Foursquare NYC) show an Imbalance Ratio (IR) of 3.75, 5.56, and 2.62.

To solve a trajectory classification problem, we collect trajectory data from GPS, LBSN, or mobile apps. This data is preprocessed to remove noise, normalize values, and segment trajectories. Feature engineering can be applied to extract relevant attributes like speed and distance. Various supervised approaches can be used, including machine learning, statistical, and deep learning models. The chosen model is then trained and optimized, with performance evaluated using metrics like accuracy and F1-score and validated through cross-validation. Finally, the model is deployed in the target application and continuously monitored for accuracy, enabling precise trajectory classification for transportation, urban planning, and personalized services.

We claim that we classify sub-trajectories since our training set is derived from the segmentation of trajectories. For brevity, hereinafter, we will use the term *trajectory classification* instead of *sub-trajectory classification*. Therefore, we use and set machine learning and deep learning algorithms to build models for predicting and assigning such labels to every sub-trajectory.

Contemporary approaches for tackling the TUL problem are generally categorized into two primary streams: (i) conventional machine learning algorithms (LEE *et al.*, 2008; ZHENG *et al.*, 2008; PATTERSON *et al.*, 2003; PATEL, 2013; BOLBOL *et al.*, 2012; TRAGOPOULOU *et al.*, 2014; VARLAMIS, 2015; FANG *et al.*, 2016), and (ii) deep learning-based methods (GAO *et al.*, 2017; ZHOU *et al.*, 2018; ZHOU *et al.*, 2019; May Petry *et al.*, 2020; FREITAS *et al.*, 2021b; FREITAS *et al.*, 2021a; FREITAS *et al.*, 2021c; CHEN *et al.*, 2022; CHEN *et al.*, 2024).

The conventional machine learning algorithms incorporate time-tested techniques for analyzing trajectory similarity, leveraging statistical models and algorithms designed to identify and quantify the resemblances and differences between known and unknown trajectories, such as Dynamic Time Warping (BERNDT; CLIFFORD, 1994). Other machine learning approaches,

such as Random Forest and XGBoost (FREITAS *et al.*, 2021c), require a feature extraction process from trajectories, such as speed, direction, and stop points, to build predictive models that accurately associate unknown trajectories with labels based on their movement patterns.

Deep learning-based methods enhance the resolution of the Trajectory-User Linking (TUL) problem by harnessing the capability to model trajectories through the extraction of rich, low-level spatial-temporal embedding from points of interest (POIs) (GAO *et al.*, 2017). These methods delve beyond simple data points to learn intricate patterns and relationships within the spatial-temporal features of user movements. By exploring the nuanced dynamics observed in user trajectories, deep learning models effectively link each trajectory to its corresponding user, leveraging the unique spatial-temporal signatures generated by each individual's movements. This approach enables a more profound and nuanced understanding of movement patterns, significantly improving the accuracy of user-trajectory associations.

In literature, deep learning-based methods such as TULER, TULVAE, and MARC have been proposed to link anonymous trajectories to their users (GAO *et al.*, 2017; ZHAO; TANG, 2017; May Petry *et al.*, 2020) for identifying and linking many check-in trajectories to their generating users using recurrent neural network (RNN). TULER and TULVAE models are limited since they only deal with a spatial feature and do not consider other essential dimensions for classification (e.g., time, venue categories, and weather conditions). Trajectory databases have become more complex, with an increase in the popularity of LBSNs. Usually, a trajectory contains spatial, temporal, and other semantic features, such as points of interest and weather conditions, among other information.

MARC (May Petry *et al.*, 2020) tackles trajectory classification problems for multidimensional data, focusing on spatial, temporal, and semantic attributes that characterize trajectories with multiple dimensions. MARC uses a multi-attribute embedding layer to encode these heterogeneous dimensions. MARC outperforms BITULER, TULVAE, and other models in (May Petry *et al.*, 2020). However, MARC does not handle imbalanced datasets and uses GeoHash for spatial representation. The output of Geohash is a binary vector linked to a dense layer.

In this work, we explore the TUL problem within multidimensional trajectory and imbalanced datasets using LBSN datasets, aiming to accurately categorize unknown subtrajectories to their respective users. We introduced a novel deep learning model named DeepeST (Deep Learning for Sub-Trajectory Classification), which employs embedding vectors inspired

by natural language processing techniques to manage large data volumes and tackle sparsity. To our knowledge, DeepeST is the first model specifically designed to address the challenges posed by imbalanced datasets for the trajectory user-linking problem (FREITAS *et al.*, 2021a). We conducted three case studies to evaluate our DeepeST's performance, comparing it against conventional machine learning algorithms such as Random Forest and XGBoost and state-of-the-art deep learning approaches including MARC, BITULER, and TULVAE, using GPS and LBSN datasets for trajectory-user linking and criminal activities. Finally, we evaluate different strategies to deal with the imbalance data problem. DeepeST achieved accuracy, precision, recall, and F1-macro values above 95% in the Gowalla, Brightkite, and Foursquare datasets.

In this work, we provide the following contributions:

1. We present an approach to classify subtrajectories using machine learning and deep learning methods from GPS services and LBSN datasets, detailed at the International Conference on Agents and Artificial Intelligence (ICAART) (FREITAS *et al.*, 2021c).
2. We propose a new deep learning model called DeepeST for trajectory classification from multidimensional extracted of GPS services and LSBN, detailed at the ICAART (FREITAS *et al.*, 2021c).
3. We evaluate ways to represent spatial and temporal features from trajectories presented at the International Florida Artificial Intelligence Research Society Conference (FLAIRS) (FREITAS *et al.*, 2021a).
4. DeepeST is the first deep learning model that provides resources for trajectory user-linking problems from imbalanced datasets, as presented in FLAIRS (FREITAS *et al.*, 2021a).
5. We perform three extensive case studies over four real-world datasets and demonstrate DeepeST's superiority over state-of-the-art approaches from machine learning and deep learning.
6. We implement an architecture and use DeepeST for trajectory classification, as presented at the International Conference on Mobile Data Management (MDM)(FREITAS *et al.*, 2021b).
7. We make our source code publicly available[1] to encourage the reproducibility of our work.
8. We implement and share diverse resources to process, analyze, and visualize trajectory data from PyMove [2].

The remainder of the thesis is organized as follows: Chapter 2 presents theoretical

[1] https://github.com/nickssonarrais/ICAART2021
[2] https://github.com/InsightLab/PyMove

fundamentals about trajectory data mining. Chapter 3 details the DeepeST Model. Chapter 4 describes two case studies, presenting and discussing the results. Finally, Chapter 5 presents considerations, future works, and a timeline.

2 THEORETICAL FUNDAMENTALS

In this chapter, we present the theoretical fundamentals related to this thesis proposal. First, we present works concerning trajectory in Section 2.1 and the techniques for trajectory processing in Section 2.2. Then, we give basic foundations about the Trajectory Classification problem in Section 2.3.

2.1 Trajectory

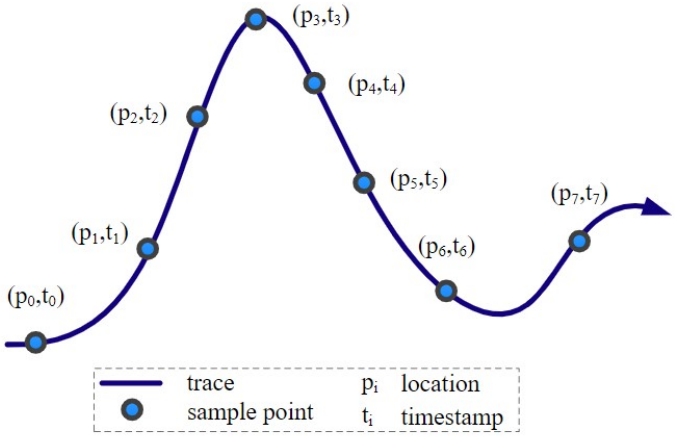
A point p is a tuple (x, y, t, A), where x and y represent the spatial coordinates (latitude and longitude, respectively) of a location and t is the timestamp of when the point was collected (BOGORNY *et al.*, 2014). The set A encompasses supplementary information directly associated with a single point or a sequence of points. Such information might include various trajectory attributes, such as velocity, acceleration, and temporal markers (e.g., day of the week, month, hour, holidays), as well as details about places visited by users, such as venue categories and weather conditions (WANG *et al.*, 2021). It is important to note that if a trajectory lacks additional data, then $A_i = \emptyset$.

A trajectory T is defined as a set of points organized in chronological order, representing the movement of objects across geographical space (ZHENG, 2015). More precisely, a trajectory is a sequential array of spatio-temporal points $[p_1, p_2, \ldots, p_m]$, arranged by time, as illustrated in Figure 2. The trajectory's length is indicated by the number of points m it contains, while its sampling rate is defined by the frequency at which these points are generated, typically measured in samples per second or other appropriate time units.

Although the definitions of trajectory and multivariate time series may seem similar, they differ in the nature of the data and their applications. A trajectory refers to a set of points describing the movement or position of an object over time in a multidimensional space and is commonly used in contexts such as object tracking, motion analysis, and geolocation (ZHENG, 2015). In contrast, a multivariate time series consists of multiple variables recorded over time, used in economic, financial, meteorological, and biomedical analyses (WEI, 2018). Trajectories focus on capturing position and movement through space over time, including both the direction and speed of movement, and directly incorporate spatial and temporal components. Multivariate time series focuses on capturing and analyzing the interdependence and relationships between multiple variables over time and may or may not have an explicit spatial component (WEI, 2018).

The analysis of trajectories often involves movement tracking techniques, spatial dynamics modeling, and movement pattern analysis, while multivariate time series analysis involves statistical techniques, correlation analysis, Fourier analysis, and predictive models (WEI, 2018). In summary, trajectories focus more on space movement over time, whereas multivariate time series deal with multiple variables changing over time.

Figure 2 – Trajectory of a moving object



Source: Feng e Zhu (2016)

A trajectory can be divided into different segments, commonly referred to as sub-trajectories. While definitions vary, generally in the literature, a subtrajectory is defined as a segment of the overall trajectory (BOGORNY *et al.*, 2014). More formally, a subtrajectory s of T consists of a list of consecutive points $[(p_k, tid_k), (p_{k+1}, tid_{k+1}), \ldots, (p_{k+l}, tid_{k+l})]$ where each p_i is part of T, with $k \geq 1$ and $k+l \leq m$ (BOGORNY *et al.*, 2014). Here, tid denotes a unique identifier for the subtrajectory.

For example, Figure 3 illustrates a sample containing two subtrajectories for a user identified as John. Each subtrajectory includes spatial coordinates (latitude and longitude), a timestamp (datetime), supplementary information associated with each point (points of interest [poi], distance to previous point [dist_to_prev], time to previous point [time_to_prev], speed to previous point [speed_to_prev]), and the subtrajectory identifier (tid).

2.1.1 Importance of Trajectory Data

The significance of trajectory data has surged with the proliferation of location-aware devices and big data technologies, enabling the collection, storage, and processing of large volumes of spatial-temporal data. Vast quantities of spatial trajectory data are gathered daily

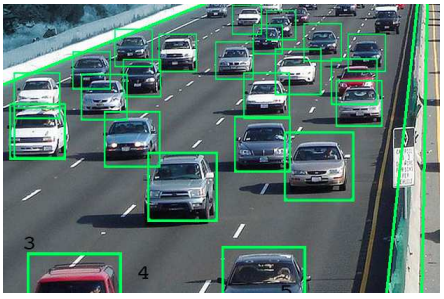
Figure 3 – Example of a subtrajectory database

	lat	lon	datetime	id	poi	dist_to_prev	time_to_prev	speed_to_prev	tid
0	39.684997	-104.940162	2010-08-19 01:42:26	John	015fe64c8896dedc051525e1f6575a2d	NaN	NaN	NaN	1
1	39.743336	-104.969148	2010-08-19 03:02:25	John	0397002b74d5c5ebddfd3ea242554782	6944.660033	4799.0	1.447106	1
2	39.758066	-104.902431	2010-08-22 17:28:50	John	369c8dbb4f1ddc861b2442cbd183182e	5934.182477	311185.0	0.019070	1
3	39.876148	-105.096502	2010-08-22 19:19:47	John	6ccba23d2e036ef1bcff1feb17443d6c7ef8f579	21145.620053	6657.0	3.176449	1
4	39.891077	-105.068532	2010-08-22 23:29:54	John	dd7cd3d264c2d063832db506fba8bf79	2907.113121	15007.0	0.193717	1
5	39.753710	-104.995135	2010-08-23 23:44:39	John	b34c1da10e5ed07bf321b5ab839aed397401720e	16510.636565	87285.0	0.189158	2
6	39.684997	-104.940162	2010-08-24 00:56:41	John	015fe64c8896dedc051525e1f6575a2d	8971.329690	4322.0	2.075736	2
7	39.891077	-105.068532	2010-08-25 02:11:29	John	dd7cd3d264c2d063832db506fba8bf79	25404.845684	90888.0	0.279518	2
8	39.891383	-105.070814	2010-08-25 04:31:35	John	7a0f88982aa015062b95e3b4843f9ca2	197.641561	8406.0	0.023512	2
9	39.740384	-104.929457	2010-08-26 00:29:24	John	8f060f74f59a02df0993b24964334eea	20680.365797	71869.0	0.287751	2
10	39.726785	-104.984852	2010-08-27 18:09:06	John	e2460b3f5040fc666cd99128a0c133b7	4972.421099	149982.0	0.033153	2

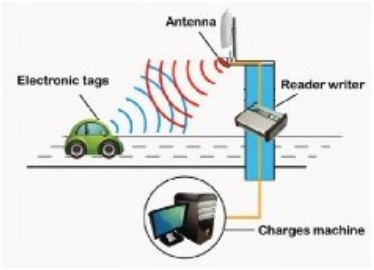
Source: Created by the author

through a range of technologies, including Global Positioning Systems (GPS), Radio Frequency Identification (RFID), smartphone sensors, 802.11 location estimation, traffic surveillance cameras, unmanned aerial vehicles (UAV), and Radio-frequency Identification (RFID), as illustrated in Figure 4.

Figure 4 – Technologies to collect trajectory data

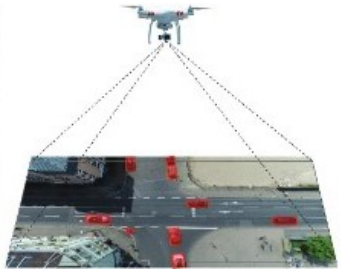


(a) Traffic surveillance cameras



(c) RFID

(b) GPS and smartphone sensors



(d) UAV

Source: adapted from Feng e Zhu (2016)

The abundant availability of trajectory data unlocks remarkable opportunities for analyzing and understanding the movement patterns and behaviors of many entities, including people, vehicles, animals, different transportation modes, and even atmospheric phenomena

like hurricanes (LEITE *et al.*, 2019). By extracting meaningful insights from these trajectories, significant advantages can be gained for individuals, businesses, and governmental organizations (BIAN *et al.*, 2019). As a result, trajectory data mining has emerged as a pivotal area of research, attracting interest from various fields. This interdisciplinary domain enables numerous applications that enhance innovation in route optimization, predictive location modeling, individual or collective movement patterns analysis, in-depth interpretation of trajectory data, and advancements in urban infrastructure and services (ZHENG, 2015). Thus, trajectory data is indispensable in supporting a broad spectrum of applications across diverse domains, including:

1. Public Safety and Security: Analyzing the movement patterns of individuals or objects enhances security surveillance systems, aids in disaster response, and improves public safety by identifying potential risks or anomalies (FREITAS *et al.*, 2021b; FREITAS *et al.*, 2021c).
2. Commercial Sector: Trajectory data enables a wide range of location-based services, from personalized recommendations, such as nearby restaurants, to dynamic pricing models in ride-sharing services (WEI; ZHANG, 2020; CHANG *et al.*, 2023; CANTURK *et al.*, 2023).
3. Urban Planning and Traffic Management: Trajectory data from vehicles and public transportation systems aids in analyzing traffic flow, identifying congestion patterns, and optimizing routes. This significantly contributes to thoughtful urban planning and reduces traffic congestion (KHAN *et al.*, 2020).
4. Environmental Monitoring: Tracking animal movements via GPS provides invaluable data for studying wildlife patterns, habitat usage, and the effects of climate change on migration. This information is crucial for conservation efforts and understanding ecological dynamics (WEI *et al.*, 2022).
5. Healthcare and Disease Control: Tracking the movement patterns of individuals within a population can model the spread of diseases, aiding in understanding disease transmission and the effective planning of control strategies (KIM *et al.*, 2020).
6. Transportation and Logistics: Analyzing the movement of vehicles enhances fleet operations, maintenance scheduling, and service reliability. Additionally, in logistics and delivery services, trajectory data mining optimizes routes, thereby reducing fuel consumption and delivery times (YANG, 2023).
7. Sports and Fitness: Athletes and fitness enthusiasts utilize trajectory data to monitor their

movements, analyze performance, and improve training regimens (BREFELD *et al.*, 2019).

These applications demonstrate the vast potential of trajectory data mining to improve operational efficiency, enhance decision-making, and contribute to societal well-being. As technology continues to evolve, the uses of trajectory data will expand, driving innovation in data analysis, machine learning, and beyond.

2.1.2 Challenges in Trajectory Data Mining

While trajectory data mining has enormous potential for a variety of applications, it also poses a distinct challenge. These issues arise from the complexity of trajectory data, which is intrinsically high-dimensional, spatiotemporal, and often vast in scale (ZHENG, 2015). Addressing these issues is critical for effectively interpreting and utilizing trajectory data.

1. **Data Volume and Complexity**: the large volume of data generated by location-aware devices can be overwhelming, leading to storage, processing, and analysis challenges. Moreover, trajectory data is multidimensional, combining spatial, temporal, and possibly other types of information, which adds to its complexity.
2. **Data Quality**: Trajectories may be sampled at irregular intervals, leading to challenges in standardizing and interpreting the data. Noise and inaccuracies in GPS data or other location technologies can significantly affect the quality of trajectory data. Dealing with these inaccuracies requires sophisticated noise reduction and data-cleaning techniques.
3. **Privacy and Security**: Trajectory data can be sensitive, as it can reveal personal locations and patterns. Ensuring privacy while mining trajectory data is a significant challenge, necessitating advanced anonymization and data protection techniques.
4. **Heterogeneity and Integration**: Trajectory data collected from different sources or devices can vary significantly in format, accuracy, and sampling rates. Integrating trajectory data with other data types (e.g., social media) is a considerable challenge for enriched analyses that require sophisticated data fusion techniques.
5. **Scalability and Real-time Processing**: As the volume of trajectory data grows, scalable data mining algorithms and infrastructure are required to process data efficiently. Besides that, there is increasing demand for real-time or near-real-time trajectory data analysis for applications such as traffic management and emergency response, which requires highly efficient processing algorithms and systems.
6. **Complexity of Spatial-Temporal Patterns**: Identifying meaningful patterns, trends, and

anomalies in trajectory data involves complex spatial-temporal analysis techniques. The dynamic nature of spatial-temporal patterns, where relationships and patterns can change over time, adds to the analysis's complexity.

Efforts to address these challenges involve interdisciplinary research, drawing from computer science, statistics, geography, and privacy law. Advances in data preprocessing, noise reduction, scalable computing, machine learning, and privacy-preserving techniques are essential for overcoming these obstacles. Collaborative efforts among academia, industry, and government agencies are crucial for developing standards and best practices for trajectory data mining (ZHENG, 2015). Overcoming these challenges will unlock the full potential of trajectory data, enabling its transformative impact across a wide range of applications (BIAN *et al.*, 2019).

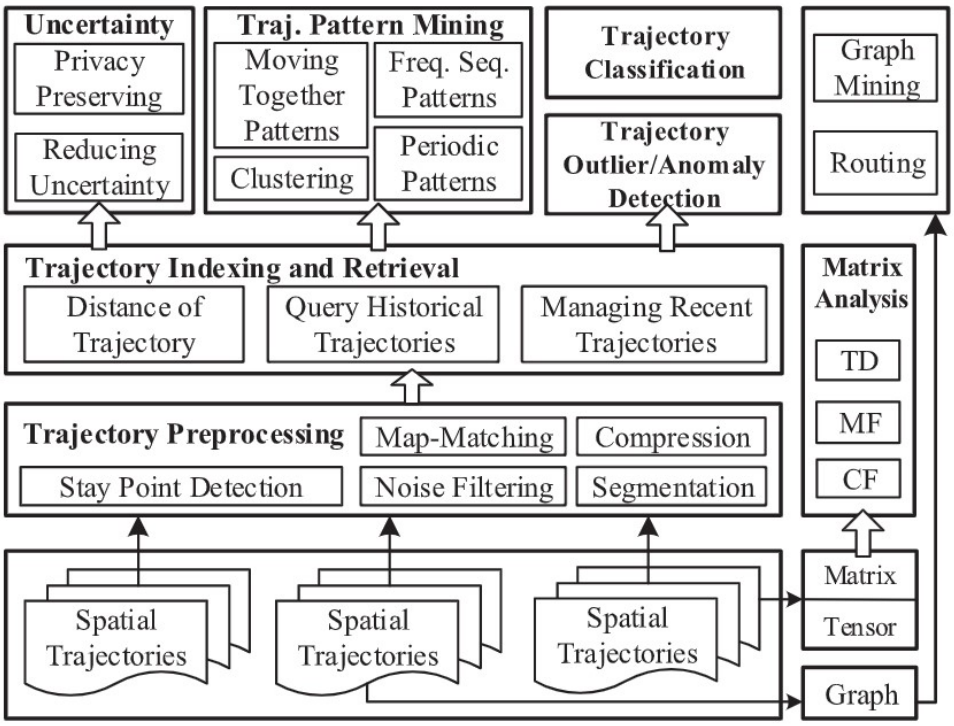
2.1.3 Paradigm of Trajectory Data Mining

In the literature, specialists and researchers have conducted individual research on trajectory data mining. Many techniques, algorithms, and methods have been proposed for processing, analyzing, managing, and mining trajectories in the past decade. Zheng (2015) provides an exhaustive review of trajectory data mining and establishes a framework that elucidates the distinctions among the existing techniques and approaches within this research area, as depicted in Figure 5. In summary, we apply methods to transform trajectories into other data formats, such as graphs, matrices, and tensors (ZHENG, 2015). We need to address issues such as noise filtering, stay point detection, segmentation, compression, and map matching during the trajectory preprocessing stage for various reasons. After this, we can apply data management algorithms and trajectory indexing methods retrieval from a trajectory to quickly retrieve particular trajectories satisfying specific criteria. Finally, there exist several research problems for trajectory data (ZHENG, 2015):

- **Trajectory Uncertainty:** A moving object's trajectory is collected at specific time intervals (e.g., days, hours, minutes, or seconds). During these intervals, the object's movement becomes unknown or uncertain, leaving us without a clear understanding of how a user travels between two points of a trajectory. A field of study has developed methods and solutions to address trajectory uncertainties (ZHENG, 2015).
- **Trajectory Pattern Mining:** This problem focuses on discovering patterns from a single trajectory or a group of trajectories. Method categories include moving together patterns, trajectory clustering, sequential patterns, and periodic patterns (ZHENG, 2015).

– **Trajectory Outlier Detection:** Using a similarity metric, we can analyze the relationship between a set of trajectories or subtrajectories to determine if an observation significantly deviates from the expected patterns or the set of trajectories (ZHENG, 2015).
– **Trajectory Classification:** Employing supervised learning approaches, we can classify trajectories or subtrajectory into various categories, such as activities (like hiking or dining) or different transportation modes, such as walking and driving (ZHENG, 2015).

Figure 5 – Paradigm of trajectory data mining



Source: Zheng (2015)

2.2 Trajectory Data Processing

Data cleaning and noise reduction are critical preprocessing steps in trajectory data mining, significantly affecting the accuracy of subsequent analyses, including classification, clustering, and pattern recognition (BIAN *et al.*, 2019). By applying a combination of techniques tailored to the specific characteristics and challenges of trajectory data, researchers and practitioners can enhance the quality and reliability of their analyses, leading to more insightful and actionable findings (ZHENG, 2015).

These processes enhance the quality of the data, ensuring that subsequent analyses are based on accurate and reliable information. Trajectory data, often collected through GPS

devices, mobile phones, or other location-aware technologies, can be noisy and inaccurate due to device errors, signal blockage, and environmental conditions.

2.2.1 Noise Filtering

Noise filtering is a crucial preprocessing step in trajectory data analysis aimed at improving data quality by reducing errors and irregularities without significantly distorting the underlying movement patterns. Trajectory data, particularly those collected from GPS devices or other sensors, is susceptible to various types of noise due to signal loss, multipath propagation (where signals bounce off surfaces before reaching the receiver), and atmospheric conditions (ZHENG, 2015). Effective noise filtering techniques, such as movement pattern recognition, trajectory classification, and clustering, are essential for accurate subsequent analysis. Traditional types of noise filtering techniques are presented below:

1. **Moving Average Filter:** This filter smooths the trajectory by replacing each point's location with the average of its surroundings within a specified window size. The window can slide along the trajectory to apply the averaging operation across the dataset. Consider a trajectory represented by a series of GPS coordinates logged every second (ZHENG, 2015). If the GPS signal momentarily loses accuracy due to interference, leading to several points deviating from the expected path, a moving average filter with a window of 5 seconds can smooth these deviations by averaging each point with two preceding and two following points.
2. **Gaussian Filter:** Similar to the moving average filter but weights the points within the window according to a Gaussian distribution, giving more importance to points closer to the center of the window. This approach often results in smoother trajectories compared to a simple moving average, especially in handling random fluctuations (or jitter). For a trajectory with irregular jitter caused by minor GPS signal fluctuations, applying a Gaussian filter can smooth the path by emphasizing the central values in the window, thereby reducing the impact of the jitter on the trajectory's overall shape (ZHENG, 2015).
3. **Median Filter:** This filter replaces each point with the median of its neighbors within a specified window rather than the mean. The median filter is particularly effective at removing outliers without affecting the rest of the data as much as averaging does, preserving sharp edges (sudden changes in direction) in the trajectory. If a trajectory includes a sudden, uncharacteristic spike in position—possibly due to a temporary signal

error—a median filter with an appropriate window size can eliminate this outlier by replacing it with the median value of its neighboring points, which are likely to represent the actual path more accurately (ZHENG, 2015).

Selecting the optimal noise filtering technique is critical and depends on the trajectory's characteristics and the analysis's specific goals (ZHENG, 2015). Key considerations include the severity and type of noise, the necessity of maintaining detailed spatial and temporal integrity, and the computational resources at hand. In practice, it might be necessary to trial various filters and parameters to determine the most effective approach for a given dataset.

2.2.2 Outlier Detection and Removal

Outliers are data points that significantly differ from the overall pattern, possibly due to measurement errors or anomalies in movement. Outlier detection and removal is a crucial step in preprocessing trajectory data, aimed at identifying and excluding data points that significantly deviate from the typical movement patterns. These outliers can arise for various reasons, including measurement errors, signal loss, or unusual behavior by the tracked object. Removing outliers is essential for improving the quality and reliability of trajectory data analysis, as they can skew results and lead to incorrect conclusions.

1. **Statistical Methods:** You can compute the Z-score for each point based on a specific attribute, such as speed or distance from the previous point, and identify outliers as those with Z-scores exceeding a predefined threshold. Alternatively, you can use the Interquartile Range (IQR) to detect outliers by examining the distribution of distances or speeds. Points that lie over 2.5 multiplied by the median or IQR far above the third or far below the first quartile are considered outliers.
2. **Distance-Based Methods:** Employ DBSCAN or other clustering algorithms to identify dense clusters of points (DENG, 2020). Points that do not belong to any cluster can be considered outliers.
3. **Speed and Acceleration Analysis:** Analyze the speeds and accelerations calculated between consecutive points. Points that result in speeds or accelerations beyond realistic limits for the context (e.g., pedestrians walking at speeds above 20 meters per second) can be identified as outliers.

2.2.3 Data Interpolation for Missing Data

Missing data in trajectory datasets can occur for various reasons, including signal loss, device malfunctions, or gaps in data collection protocols. These gaps can disrupt the analysis of movement patterns, affect the calculation of speed and distance, and impair the application of algorithms that require continuous data sequences. Interpolation helps overcome these challenges by filling in the gaps, allowing for a more accurate trajectory representation.

Data interpolation in the context of trajectory data refers to estimating missing or unrecorded data points within a sequence of spatial-temporal points. This step is crucial for maintaining the continuity and integrity of trajectory datasets, as missing data can lead to inaccurate analyses and interpretations of movement patterns. Some interpolation methods are detailed below:

1. **Linear Interpolation:** The simplest form of interpolation estimates missing points by directly connecting the points before and after the gap with a straight line. The assumption is that the object moved constantly and directly between the two known points (ZHENG, 2015). The main disadvantage of linear interpolation is that it may not accurately capture natural variations in object movement, especially in trajectories involving curves or changes in speed. For example, consider a dataset tracking vehicle on a highway with missing points due to a tunnel where GPS signals were lost. Linear interpolation might be used to estimate the positions of the cars inside the tunnel, assuming constant speed and direction through the short gap. However, linear interpolation might be inaccurate if the tunnel includes curves.
2. **Spline Interpolation:** offers significant advantages for more complex trajectories. This method uses piecewise polynomials to connect known points, allowing for a smoother and more flexible estimate of positions (ZHENG, 2015). Splines are particularly useful in scenarios where the trajectory is curvilinear, or the object's speed varies along the path. Thus, spline interpolation can more accurately capture changes in movement and direction, providing a more realistic and continuous representation of the object's path. This characteristic makes spline interpolation ideal for contexts where path accuracy is crucial, such as in analyzing the movements of vehicles on winding roads or tracking animals that move irregularly and rapidly. However, one significant disadvantage of spline interpolation is its potential to overfit the data, especially in scenarios with sparse or noisy data points.

3. **Kinematic Interpolation**:This approach incorporates knowledge about the object's motion to estimate missing points such as acceleration or deceleration patterns. Kinematic interpolation is a more complex method but can provide more accurate estimations for objects whose movement is influenced by forces. In tracking bird migration, data might be missing due to the birds flying through areas with poor signal reception (ZHENG, 2015). Since birds can change altitude and direction rapidly, it could be used to estimate the missing points, considering their potential acceleration and curvilinear paths. However, compared to more straightforward methods such as linear or spline interpolation, kinematic interpolation is more complex regarding the required calculations. This can make it impractical in situations that require real-time processing or when computational resources are limited.

2.2.4 Trajectory Segmentation

Trajectory segmentation involves dividing a continuous trajectory into meaningful segments or subtrajectories based on specific criteria or characteristics (DAMIANI; HACHEM, 2017). This process is crucial in trajectory data mining, as it helps identify patterns, understand behavior, and facilitate more detailed analyses. Segmentation criteria vary widely, including speed, direction, stop points, or contextual information like changes in surrounding environments or activities.

Trajectory segmentation is important for several reasons, as it significantly enhances the analysis and interpretation of movement data across various fields and applications. Here are some key reasons why trajectory segmentation is crucial:

- **Enhanced Data Analysis:** Segmentation allows for the analysis of specific trajectory parts, making it possible to identify patterns or anomalies that might be obscured when looking at the trajectory as a whole.
- **Behavioral Insight:** Researchers can gain insights into the behavior of moving objects, whether they are humans, animals, or vehicles by segmenting trajectories based on behavior, such as stopping, turning, or accelerating.
- **Efficiency in Processing**: Segmentation can reduce the complexity of data, making it easier and more efficient to process and analyze large datasets.

For example, we can segment the daily trajectories of commuters based on their stops (e.g., home, workplace, schools) (ZHENG, 2015). This segmentation can help understand

commuting patterns, peak travel times, and public vs. private transportation. We can divide the migratory paths of birds into segments based on changes in direction or stopover points. Such segmentation can reveal important resting spots, changes in migration routes due to environmental factors, and patterns in flight behavior. In sports analytics, an athlete's movement data during a game can be segmented based on high-activity (sprints) and low-activity (rests) periods. This segmentation can provide insights into the athlete's endurance, performance dynamics, and strategy.

There are some approaches to trajectory segmentation (ZHENG, 2015):

- Time-based Segmentation: Dividing a trajectory into segments based on fixed or variable time intervals.
- Distance-based Segmentation: Segmenting based on the distance traveled is useful for analyzing movement patterns over specific distances.
- Speed/Direction Change: Identifying segments based on changes in speed or direction, indicating changes in behavior or activity.
- Stoppage Detection: involves segmenting a trajectory by identifying periods of immobility or low movement, often referred to as stops or dwell times. This technique is particularly useful for understanding the usage patterns of specific locations and can provide insights into the behavior or activity of the tracked entity.

The segmentation techniques outlined thus far encapsulate the primary methodologies employed in trajectory analysis. However, there exist additional, although less frequently utilized, segmentation strategies that could afford insights into movement dynamics. One such strategy is user-defined segmentation, which empowers analysts to customize segmentation criteria to align with specific investigative queries or objectives (ZHENG, 2015). This approach introduces flexibility that is particularly advantageous in examining intricate datasets. Moreover, multidimensional segmentation engages multiple attributes in concert, such as speed, direction, and contextual variables, to delineate segments that expose more nuanced and complex movement patterns. While these advanced segmentation methodologies are less widely adopted, they pave the way for a more comprehensive analysis and interpretation of trajectory data, suggesting their significant potential to contribute to movement pattern analysis.

2.2.5 Map-maching

Map matching is a process used in geographic information systems (GIS) to align a sequence of observed geographic positions (e.g., GPS data points) with the road network on a digital map (HUANG *et al.*, 2021). This technique is essential for converting raw GPS data, which can be noisy and imprecise, into a more accurate representation of an object's movement along the transportation network. Map matching improves the reliability of trajectory data by correcting deviations due to GPS errors, signal reflections, or inaccuracies in position reporting.

Map matching algorithms take GPS trajectories as input and output a sequence of map features (e.g., road segments) that best match the observed movements. These algorithms consider various factors, including the geometry of the road network, the direction of travel, and the speed, to accurately associate GPS points with the correct paths on the map. There are several approaches to map matching, ranging from simple geometric methods to more complex ones that use probabilistic models or machine learning (CHAO *et al.*, 2020; HUANG *et al.*, 2021).

For example, consider a delivery truck with a GPS tracker navigating through an urban area generating GPS data points every few seconds. Due to the dense building environment, some GPS signals reflect off buildings, leading to inaccurate positions that do not align with the road network. Using map matching, these points can be corrected to fit the truck's most probable route based on the available road network data and the sequence of recorded points (CHAO *et al.*, 2020). For instance, if the GPS data erroneously places the truck on the wrong side of a river with no nearby bridge, map matching algorithms can correct this by aligning the truck's path with the nearest bridge and the roads it actually could have used, given its starting point and destination.

2.3 Trajectory Classification

We can use supervised learning approaches to classify trajectories or segments of a trajectory into some categories. The trajectory classification problem involves a prediction model to classify a new trajectory in a single class (like hiking and dining) or multi-class (car, bus, train, etc.). We can use conventional machine learning algorithms or deep learning approaches to create models that learn the patterns (or classes) from historically labeled trajectory (or sub-trajectory) data (FREITAS *et al.*, 2021c). Basic examples of trajectory classification are:

(i) determining the transportation mode of the moving object like a car, bus, bike, taxi,

airplane, or train (ZHENG *et al.*, 2008; BOLBOL *et al.*, 2012; TRAGOPOULOU *et al.*, 2014; VARLAMIS, 2015).

(ii) classify animal categories such as rabbit, dog, and leopard (LEE *et al.*, 2008).

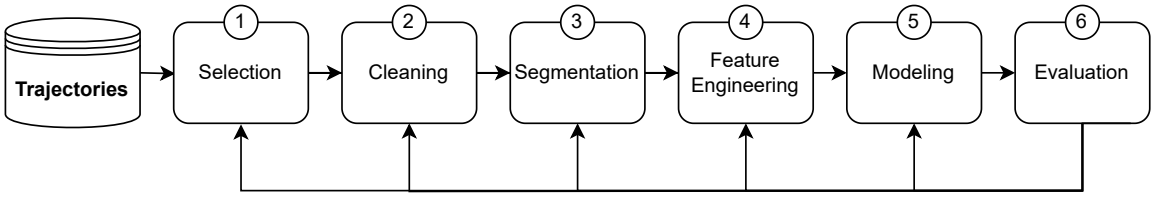
(iii) determines a user's next position like home, school, cafe, office, or restaurant (FREITAS *et al.*, 2021b; FREITAS *et al.*, 2021a).

(iv) identifying the user of a trajectory (GAO *et al.*, 2017; ZHOU *et al.*, 2018; May Petry *et al.*, 2020).

We claim that we classify sub-trajectories since our training set is derived from the segmentation of trajectories. For brevity, hereinafter, we will use the term *trajectory classification* instead of *sub-trajectory classification*. Therefore, we use and set machine learning and deep learning algorithms to build models for predicting and assigning such labels to every sub-trajectory.

The sub-trajectory classification process generally involves six main steps, as illustrated in Figure 6. Initially, we select relevant trajectories to develop a classification model from a trajectory database. Trajectories often experience noise, inconsistency, and incompleteness for various reasons. Consequently, we address noise, identify points of inconsistency, and even compress trajectories during the preprocessing stage (ZHENG, 2015), as depicted in Figure 6.

Figure 6 – Main steps to subtrajectory classification



Source: Created by the author

In some scenarios, such as classification, we can split a trajectory into subtrajectories in the segmentation step according to the behaviors of moving objects. Each sub-trajectory is often called a segment, a partition, or a frame. We segment trajectory to efficiently store sample points of a moving object aligned by time intervals, leveraging a state-of-the-art column-oriented storage system. The segmentation reduces the computational complexity and supports richer knowledge of sub-trajectory patterns from trajectories. Several trajectory segmentation methods are based on a trajectory's shape, time interval, and semantic meanings (ZHENG, 2015). To learn more about segmentation, you can access the subsection 2.2.4 of this document.

After segmenting the trajectories, we can extract features from the sub-trajectories,

including velocity, acceleration, days of the week, position, etc. We also can integrate semantic attributes from external databases, such as Google Maps (point of interest, visited place, price tier, and rating), meteorological databases (weather condition, temperature), and other spatiotemporal data.

In the modeling step, we use supervised learning algorithms to classify subtrajectories into various categories. This involves training a model to identify specific patterns and attributes within the trajectory data. The categories can include: (1) users that are the owner of the trajectory, (2) transportation modes (car, bus, bike, walk), (3) types of criminal activities, or any set of categorical data associated with trajectories.

There are several approaches that can be used to build classification models. In this work, we focus on the Trajectory User-Linking (TUL) problem that aims to classify trajectories to their generating users. Contemporary approaches for TUL problem are generally categorized into two primary streams: (i) conventional machine learning algorithms (LEE *et al.*, 2008; ZHENG *et al.*, 2008; PATTERSON *et al.*, 2003; PATEL, 2013; BOLBOL *et al.*, 2012; TRAGOPOULOU *et al.*, 2014; VARLAMIS, 2015; FANG *et al.*, 2016), and (ii) deep learning-based methods (GAO *et al.*, 2017; ZHOU *et al.*, 2018; ZHOU *et al.*, 2019; May Petry *et al.*, 2020; FREITAS *et al.*, 2021b; FREITAS *et al.*, 2021a; FREITAS *et al.*, 2021c; CHEN *et al.*, 2022; CHEN *et al.*, 2024).

2.3.1 Conventional Machine Learning algorithms

Conventional ML-based classification models, such as k-nearest neighbors (KNN) (SHARMA *et al.*, 2010), Support Vector Machines (SVM) (LEE *et al.*, 2008), Random Forest (FREITAS *et al.*, 2021c; FREITAS *et al.*, 2021b), and XGBoost (FREITAS *et al.*, 2021c; FREITAS *et al.*, 2021b), among others, can be employed to tackle the problem of trajectory classification by transforming trajectories into one-hot vector representations. This method encodes trajectory data into a binary format, where each element of the vector corresponds to a specific state or position within the trajectory. Each state is marked as '1' if it occurs, and '0' otherwise. Such a transformation enables conventional algorithms to process and analyze trajectory data similarly to how they would handle any other categorical or numerical dataset. For example, using this vector representation, a Random Forest and XGBoost could be trained to predict types of criminal activity (e.g., at home, blocking the signal from the equipment that transmits his/her location, selling stolen car parts, in a hearing with the judge, among other categories) based on the patterns detected in the binary vectors (FREITAS *et al.*, 2021b;

FREITAS *et al.*, 2021c). The arsenal of conventional machine learning techniques for such tasks is extensive and includes, but is not limited to:

- Decision Trees: A model that classifies data by sequentially splitting it based on feature values, forming a tree-like structure of decisions (KOTSIANTIS, 2013). Decision Trees are straightforward to understand and interpret, making them a popular choice for initial exploratory models.
- K-Nearest Neighbors (KNN): This algorithm classifies data points based on the majority class among the k nearest neighbors in the feature space. KNN is highly intuitive and requires no explicit model training phase, although it can be computationally demanding with large datasets (GUO *et al.*, 2003).
- Support Vector Machines (SVM): SVMs are powerful classifiers that find the optimal hyperplane to separate different classes in the feature space (BENNETT; DEMIRIZ, 1998). They are capable of handling both linear and non-linear classifications through the use of kernel functions, offering versatility in various application scenarios.
- Random Forest: An ensemble method that utilizes multiple decision trees to improve classification outcomes. By training on different subsets of the data and features, Random Forests aim to reduce overfitting and increase model robustness, often delivering high accuracy (BREIMAN, 2001).
- Extreme Gradient Boosting (XGBoost): A highly efficient and scalable implementation of gradient-boosted decision trees, XGBoost is renowned for its performance and speed (CHEN; GUESTRIN, 2016). It has been the algorithm of choice in numerous data science competitions and practical applications, attributed to its effectiveness in handling diverse types of data.

However, these methods fail to consider spatial-temporal features and often perform inferior to deep learning-based models, as presented in (FREITAS *et al.*, 2021c).

2.3.2 Deep Learning based methods

Deep learning-based methods offer several advantages over conventional machine learning methods for solving the classification problem from spatio-temporal data, including improved accuracy, handling high dimensional trajectory data, automatic feature representation learning, robustness to noise and outliers, and scalability (WANG *et al.*, 2022).

Recurrent Neural Network (RNN), more specifically, Long Short-Term Memory

(LSTM) is a recurrent neural network (RNN) architecture used in deep learning. LSTM processes not only single data points (such as images), but also entire data sequences (such as sub-trajectories, video, or speech). DeepeST employs an LSTM (HOCHREITER; SCHMIDHUBER, 1997), which has been extensively used to process variable-length input and can allow highly non-trivial long-distance dependencies to be easily learned.

In summary, LSTM has been used in state-of-the-art works for trajectory classification from multidimensional data due to its capacity to learn complex patterns from a sequence (GAO *et al.*, 2017; ZHENG *et al.*, 2008; May Petry *et al.*, 2020; FREITAS *et al.*, 2021a), unlike feedforward neural networks.

2.3.3 Evaluation Metrics for Trajectory Classification

Evaluation metrics for trajectory classification play a crucial role in assessing the performance and accuracy of models used to classify trajectories based on their characteristics and behaviors. These metrics help to quantify how well a classification model can differentiate between various types of trajectories, such as distinguishing between different modes of transportation (walking, driving, cycling) or different animal movement patterns.

Evaluation metrics for trajectory classification are usually extracted from the values of a confusion matrix.

- **True Positives (TP)**: the case where the true label is positive and whose class is correctly predicted to be positive.
- **True Negatives (TN)**: the case where the true label is negative and whose class is correctly predicted to be negative.
- **False Positives (FP)**: the case where the true label is negative and whose class is incorrectly predicted to be positive.
- **False Negatives (FN)**: the case where the true label is positive and whose class is incorrectly predicted to be negative.

An overview of crucial evaluation metrics commonly used in trajectory classification is presented below.

Accuracy: It is a useful traditional measure when the problem has well-balanced classes. Accuracy provides an overall performance for a model. It can be calculated by the sum of the element on the diagonal of the confusion matrix divided by the total number of predictions

made, according to the equation 2.1:

$$\textbf{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.1}$$

To evaluate precision, recall, and f1-score, we consider a macro average that computes each class's score separately. This kind of averaging is useful when the dataset is imbalanced, and we aim to concentrate the same emphasis on all the classes. Let o be the number of labels in Y, TP_i is the number of true positives of the label y_i, FP_i is the number of false positives of the label y_i, TN_i is the number of true negatives of the label y_i, and FN_i is the number of false-negatives of the label y_i.

Macro Precision: measures the ability of a classifier to identify only the correct instances for each class. Macro precision is show in equation 2.2:

$$\textbf{Precision}_{(\mathbf{M})} = \frac{\sum_{i=1}^{o} \frac{TP_i}{TP_i+FP_i}}{o} \tag{2.2}$$

Macro Recall: measures the ability of a classifier to find all correct instances per class. On the other hand, recall refers to the percentage of total relevant results correctly classified by your algorithm. The formula for macro recall is shown in equation 2.3.

$$\textbf{Recall}_{(\mathbf{M})} = \frac{\sum_{i=1}^{o} \frac{TP_i}{TP_i+FN_i}}{o} \tag{2.3}$$

Macro F1-score: is the mean of precision and recall normalized between 0 and 1. An F1 score of 1 indicates a perfect balance, as precision and recall are inversely related. A high F1 score is useful when both high recall and precision are important. The formula for macro F1-score is shown in equation 2.4.

$$\textbf{F1}-\textbf{score}_{(\mathbf{M})} = \frac{2 \cdot \textbf{Precision}_{(\mathbf{M})} \cdot \textbf{Recall}_{(\mathbf{M})}}{\textbf{Precision}_{(\mathbf{M})} + \textbf{Recall}_{(\mathbf{M})}} \tag{2.4}$$

Balanced Accuracy:Balanced accuracy is a more robust metric compared to traditional accuracy, especially when dealing with imbalanced datasets. It considers both the sensitivity (recall) and the specificity for each class, providing a more equitable evaluation by giving equal weight to each class, regardless of their frequency in the dataset. Balanced accuracy

is particularly important when one or more classes are underrepresented, as it ensures that the model's performance is fairly assessed across all classes. For binary classification, balanced accuracy is defined as the average of the sensitivity and specificity:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

For multi-class classification, balanced accuracy is calculated as the average of the balanced accuracies for each class. This can be expressed as:

$$\textbf{Balanced Accuracy} = \frac{1}{o}\sum_{i=1}^{o}\left(\frac{TP_i}{TP_i+FN_i} + \frac{TN_i}{TN_i+FP_i}\right) \tag{2.5}$$

Where o is the number of classes, TP_i represents the true positives for class y_i, FN_i represents the false negatives for class y_i, TN_i represents the true negatives for class y_i, and FP_i represents the false positives for class y_i. Balanced accuracy is essential in imbalanced datasets because it provides a better understanding of the model's performance across all classes. In such scenarios, traditional accuracy might be misleading, as it could reflect high performance simply by predicting the majority class correctly most of the time while neglecting the minority class. By considering both the recall and specificity of each class individually and then averaging them, balanced accuracy gives a more meaningful assessment of a model's capability to correctly classify all classes, making it a crucial metric for tasks where fairness across classes is vital.

Top-K Accuracy: is an essential metric for evaluating models in multi-class classification tasks, especially when dealing with a large number of classes. It measures whether the true class is within the top K predicted probabilities. This metric is particularly valuable in applications where it is important to ensure that the correct class is among the top K predictions, such as user identification and recommendation systems. The formula for Top-K accuracy is shown in equation 2.6, where $I(\cdot)$ is an indicator function that equals 1 if the condition is true and 0 otherwise, y_i is the true class, for instance, x_i, and n is the total number of instances.

$$\textbf{Top-K Accuracy} = \frac{\sum_{i=1}^{n} I(y_i \in \text{Top-K predictions for } x_i)}{n} \tag{2.6}$$

3 DEEPEST

In this chapter, we will present the first contribution of this work, which was the deep learning model called DeepeST (Deep Learning for subtrajectory Classification) for trajectory classification.

3.1 Trajectory User-Linking Overview

Trajectory classification has been a widely studied problem in trajectory pattern mining. In the beginning, trajectory classification focused on detecting patterns of mobility from raw trajectories using machine learning methods (ZHENG *et al.*, 2008; PATTERSON *et al.*, 2003; FANG *et al.*, 2016; BOLBOL *et al.*, 2012; TRAGOPOULOU *et al.*, 2014; VARLAMIS, 2015). One of the first methods for trajectory classification was TraClass, proposed by Lee in (LEE *et al.*, 2008), which supports only the spatial dimension. Patel extended the TraClass to support the spatial and time dimensions in (PATEL, 2013).

Afterward, they started to use more features to increase the performance of the machine-learning methods. Bolbol *et al.* (2012) segments the trajectories based on a pre-defined number of subtrajectories and extracts features from subtrajectories inside the sliding window of fixed size like average acceleration and average speed, and uses them as input to an approach based on Support Vector Machines (SVMs) classification. Tragopoulou *et al.* (2014) infers more features for identifying trajectory transportation modes, such as whether the day of the week is a workday, altitude, timezone, the status of GPS services, and others. Varlamis (2015) extended the previous work by adding the information based on the distance, for example, whether the point is near tourist places, whether on a bus or train line. These features allow them to distinguish between working or spending the night out between daily and weekend habits.

These works use machine learning methods demanding a feature extraction process to categorize raw trajectories or (subtrajectories) into different motion patterns considering features extracted from the spatial and temporal dimensions like velocity, acceleration, and distance.

More recently, studies involve deep learning models for the Trajectory-User Linking (TUL) problem as (GAO *et al.*, 2017; ZHOU *et al.*, 2018; ZHOU *et al.*, 2019; May Petry *et al.*, 2020; CHEN *et al.*, 2022; CHEN *et al.*, 2024), and they aim at linking anonymous trajectories to the users who generate them. The implications of TUL are vast and varied, encompassing

areas such as identity verification, personalized content delivery, epidemiological surveillance, and security threat assessment, all of which significantly benefit both commercial interests and public safety initiatives.

For instance, by linking users to their points of interest (POI) check-ins, advertisers can obtain a more nuanced understanding of consumer preferences. This insight allows for the delivery of more targeted and highly personalized advertisements, enhancing the effectiveness of marketing campaigns. In security, TUL is a powerful tool for identifying atypical movement patterns with people in criminal databases (FREITAS *et al.*, 2021b; GAO *et al.*, 2017). It makes it possible to flag potential criminal or terrorist activities, bolstering efforts to maintain public safety.

Unlike traditional trajectory classification methods that categorize trajectories or trajectory segments based on spatiotemporal values and activities into predefined categories (like walking or jogging) or transportation modes (bus, car, bike e, etc), TUL aims to link trajectories directly to users. This presents unique challenges, including:

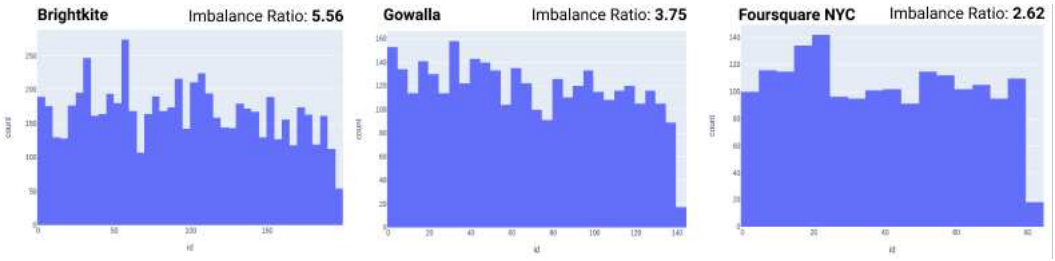
1. Trajectory databases often contend with large volumes of data, primarily due to the substantial number of users and extended data collection periods. Each user's movements contribute to a growing dataset where entries are continuously logged over time, capturing their locations at various times. For example, the Gowalla dataset[1] comprises 6,442,890 check-ins between Feb. 2009 and Oct. 2010. Similarly, the Brightkite dataset[2] contains 4,491,143 check-ins, further showcasing the immense scale of spatial and temporal data captured between Apr. 2008 and Oct. 2010.
2. The complexity of representing spatial and temporal data is significant (YANG *et al.*, 2015). First, handling and representing latitude and longitude poses challenges because these geographic coordinates are continuous and highly dimensional variables that can exhibit considerable variation over short distances, complicating the accurate capture of local spatial relationships. Furthermore, the Earth's spherical shape adds another layer of complexity in measuring distances and relationships between coordinates. The Euclidean distance metrics commonly used in standard models do not accurately represent the distances on a spherical surface. Finally, handling and representing date and time data in deep learning models poses significant challenges. Time features are inherently cyclical, such as hours that repeat every 24 hours and days that cycle annually, which

[1] https://snap.stanford.edu/data/loc-gowalla.html
[2] https://snap.stanford.edu/data/loc-Brightkite.html

can be problematic for standard neural networks that expect linear input data. Additional complexities include adjustments for time zones, daylight saving changes, and the impact of holidays and special events, which require sophisticated preprocessing to ensure accurate model predictions.

3. The sequence of spatio-temporal points is often intermittent and widely spaced in both time and location (YANG *et al.*, 2015), posing a significant challenge in accurately modeling and predicting user trajectories. This irregularity leads to gaps and inconsistencies in the data, making it difficult to capture continuous movement patterns; consequently, traditional trajectory analysis methods, which often rely on regular and dense data, may not perform well. Therefore, developing robust techniques that can handle the sporadic nature of check-ins and effectively link user trajectories becomes crucial for improving the accuracy and reliability of user classification in LBSN.
4. The nature of multiple dimensions, as trajectories, can be linked to other properties such as geographic contexts, temporal patterns, behavioral intents, and environmental factors (May Petry *et al.*, 2020).
5. The number of classes can be much larger than the number of motion patterns. For example, the Brightkite dataset contains 51,406 users, and Gowalla possesses 107,092 users. (YANG *et al.*, 2015).
6. Trajectory datasets may present imbalanced distributions of the target variable, considering an Imbalance Ratio (IR) greater than two (FERNÁNDEZ *et al.*, 2008). For instance, considering the TUL problem and the three LBSNs datasets (Gowalla, Brightkite, Foursquare NYC), some users check in more frequently than others, and the IRs of these datasets are respectively 3.75, 5.56, and 2.62, as shown in Figure 7.

Figure 7 – Number of users class by subtrajectories for three trajectory datasets



Source: Created by the author

In literature, deep learning-based method have been proposed to link trajectories to their generating users since 2017. TULER was the first deep learning model introduced in (GAO

et al., 2017) for identifying and linking many check-in trajectories to their generating users using recurrent neural network (RNN) based models. In 2018, TULVAE, also using RNN, was proposed in (ZHOU *et al.*, 2018). In 2019, improvements in TULER were presented in (ZHOU *et al.*, 2019), a generative model to mine human mobility patterns, which aims at learning the implicit hierarchical structures of trajectories and alleviating the data sparsity problem with semi-supervised learning. TULER and TULVAE models are limited since they only deal with a spatial feature and do not consider other essential dimensions for classification (e.g., time features), as shown in Table 1.

Table 1 – Main Deep Learning model for TUL problem

Model	RNN	Multidimensional Data	Spatial Representation	Imbalanced dataset
DeepeST	LSTM/BLSTM	supports all features	Grid Index linked to an embedding and GeoHash linked to a dense layer	Focal Loss/CBCE
MARC	LSTM	supports all features	GeoHash linked to a dense layer	not support
TULER	LSTM/GRU/BLSTM	supports only spatial feature	not support	not support
TULVAE	LSTM/GRU	supports only spatial feature	not support	not support

Trajectory databases have become more complex, with an increase in the popularity of LBSNs. Usually, a trajectory contains spatial, temporal, and other semantic features, such as points of interest and weather conditions, among other information.

MARC (May Petry *et al.*, 2020) tackles trajectory classification problems for multi-dimensional data, as does DeepeST. Both proposals focus on spatial, temporal, and semantic attributes that characterize trajectories with multiple dimensions. MARC uses a multi-attribute embedding layer to encode these heterogeneous dimensions. MARC outperforms BITULER, TULVAE, and other models in (May Petry *et al.*, 2020). However, MARC does not handle imbalanced datasets and uses different spatial representation approaches compared to DeepeST. MARC uses GeoHash for spatial representation, which is linked to a dense layer, and DeepeST uses Grid Index, which is connected to an embedding layer, as shown in Table 1. Beyond providing a solution for imbalanced data for trajectory classification, this work compares both models for trajectory classification from multidimensional data.

3.2 Problem Statement

Let a trajectory T be a sequence of points sorted in time $[p_1,..,p_{len_j}]$. Here, p_i $(1 \leq i \leq len_j)$ is a tuple (l_i,t_i,A_i), such that l_i is a location point at time t_i, and $A_i = [a_1,...,a_m]$ is a sequence of m attributes linked to the trajectory (e.g. velocity, acceleration, geographic information, among others). It is worth mentioning that if a trajectory is not linked to any

semantic information, then $A_i = \emptyset$. For the sake of brevity, the location point l_i is a grid index generated from spatial coordinates, e.g., latitude and longitude collected from a GPS device, or l_i can refer to check-in or stop location (it can be a POI location, for instance), while each timestamp t_i to a time slot in $T = \{t_1, t_2, ..., t_w\}$, such that $T \in \mathbb{R}^w$. A time slot could be a regular time interval, for instance, some minutes, hours, days, or weeks.

Finally, to reduce the computational complexity and capture richer knowledge of subtrajectory patterns from trajectories, we segment the trajectories into subtrajectories. There are several trajectory segmentation methods based on a trajectory's shape, time interval, and semantic meanings (ZHENG, 2015). Since trajectory segmentation is not at the core part of this work, we adopt the simplest method based on the time interval to trajectories. We are now ready to formulate our prediction problem for subtrajectories.

Given a set of subtrajectories $S = \{s_1, s_2, \ldots, s_z\}$, the task is to predict the category by linking each subtrajectory $s_i \in S$ to a label $y \in Y = \{y_1, \ldots, y_o\}$. In this work, we focus on the TUL problem, but it is important to note that our problem is generic; Y can be a set of transportation modes (car, bus, bike, or walk), a set of users that are owners of the trajectory, or any categorical feature from other domains. We define whether the dataset is imbalanced based on the Imbalance Ratio (IR). The IR quantifies the disparity in the distribution of subtrajectories across different categories. Specifically, we consider a dataset to be imbalanced if the IR exceeds 2 (FERNÁNDEZ *et al.*, 2008). This means that the number of subtrajectories in the most frequent category is more than twice the number of subtrajectories in the least frequent category. Mathematically, if we denote the total number of subtrajectories z associated with each category y_j as n_j for $j = 1, 2, \ldots, o$, then the imbalance ratio (IR) can be expressed as:

$$\text{IR} = \frac{\max(n_1, n_2, \ldots, n_o)}{\min(n_1, n_2, \ldots, n_o)}$$

Here, $n_1, n_2, \ldots, n_o$ represent the number of subtrajectories corresponding to each category $y_1, y_2, \ldots, y_o$ respectively. For example, n_1 is the number of subtrajectories labeled with category y_1, n_2 is the number of subtrajectories labeled with category y_2, and so on.

Given that IR > 2, it implies that:

$$\frac{\max(n_1, n_2, \ldots, n_o)}{\min(n_1, n_2, \ldots, n_o)} > 2$$

This inequality indicates that the category with the maximum number of subtrajectories has more than twice the quantity of subtrajectories compared to the category with the

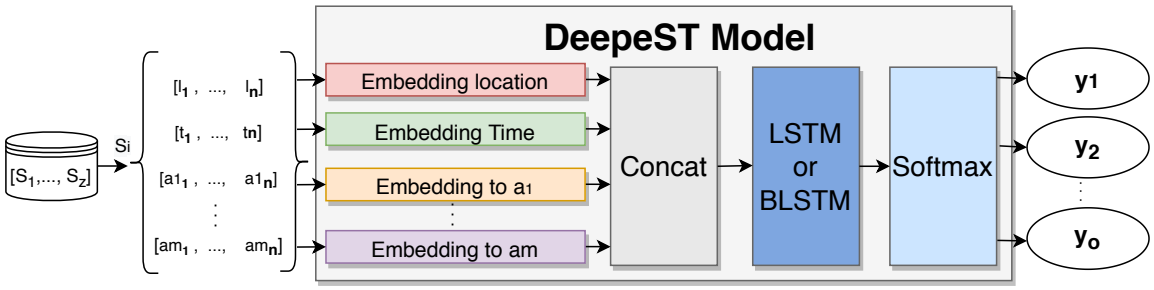
minimum number of subtrajectories. This level of imbalance can significantly affect the performance of traditional classification models, leading to biased predictions toward the majority classes. Therefore, specialized techniques such as re-sampling methods or adjusted loss functions, such as Focal Loss or Class-Balanced Loss, are often necessary to address this challenge effectively.

3.3 DeepeST Architecture

In this session, we will delve deeper into DeepeST and understand its main components for subtrajectory classification.

The DeepeST architecture comprises embedding layers to each input, a concatenation layer, a recurrent layer (LSTM or BLSTM), and a fully connected layer with a softmax activation function. The overview of DeepeST is illustrated in Figure 8.

Figure 8 – DeepeST model to subtrajectory classification



Source: Created by the author

3.3.1 Subtrajectory encoding

The first challenge to the trajectory classification problem is the data representation. The spatial dimension of trajectories is composed of two attributes: latitude and longitude. Therefore, we need to represent trajectory data for spatial and temporal features.

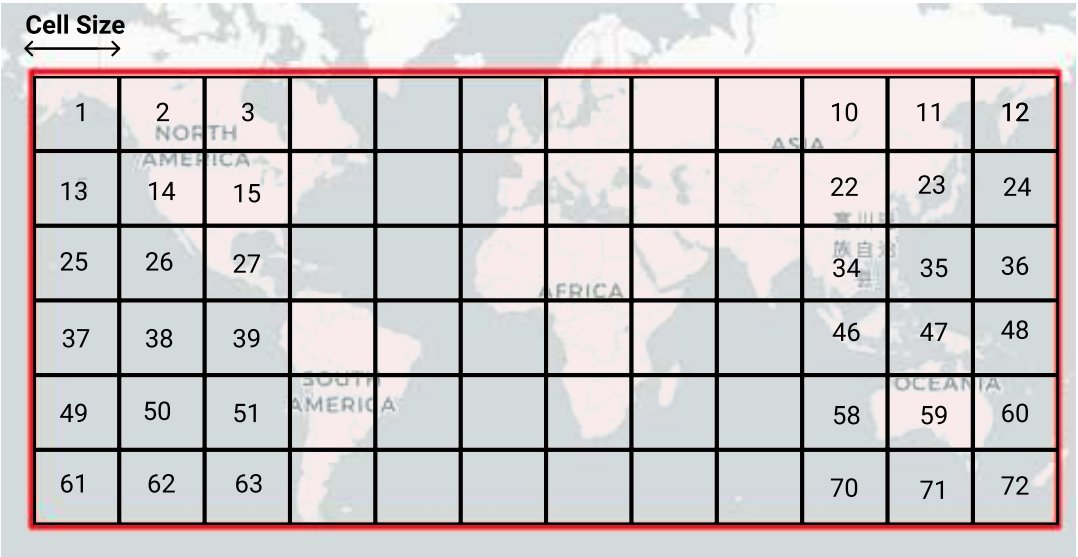
Representing latitude and longitude presents significant challenges due to the continuous and highly dimensional nature of geographic regions. Additionally, the spherical shape of the Earth introduces further complexity in accurately measuring distances between points. To address these challenges, we implemented a grid-based solution in our work.

By dividing the geographic area into a grid of cells, we can discretize the continuous space into manageable units. This approach simplifies the representation of geographic locations, making the data easier to handle and analyze. The grid-based method allows us to efficiently

process and compare spatial data, facilitating more effective trajectory analysis.

DeepeST creates a spatial grid 2D with a cell of fixed size in meters covering all points in the dataset as shown in Figure 9. More formally, we represented spatial location l_i composed of latitude and longitude from a trajectory TR, within a specific grid cell, where each cell is indexed by an integer. The user can adjust the cell size, and we experimented with various grid index parameters to assess the impact on tracking precision. A key drawback of using an integer-based spatial grid is the potential loss of precision, as fixed-size cells can lead to abrupt transitions between adjacent cells with only minimal movement near boundaries. This affects the accuracy of capturing fine-scale movements, particularly in densely populated areas. Higher cell size values (e.g., 10,000 meters) make it difficult to capture small, localized movements (such as within a parking lot). Conversely, smaller cell sizes enable the detection of transitions between geographical areas during movement but increase the overall number of grid cells, adding computational complexity.

Figure 9 – 2D Spatial Grid



Source: Created by the author

Furthermore, we employed the Haversine formula to measure distances between points on the Earth's surface accurately (ROBUSTO, 1957). The primary advantage of the Haversine formula over the Euclidean distance is its ability to provide accurate distance calculations on the spherical surface of the Earth. The Haversine formula considers the Earth's curvature, ensuring precise measurements between two points given their latitude and longitude coordinates (WINARNO *et al.*, 2017). This is particularly important for geospatial applications such as trajectory analysis, navigation, and geolocation, where accurate real-world distances are

crucial. In contrast, Euclidean distance assumes a flat plane, which can lead to significant errors over large geographic distances. The Haversine formula is especially beneficial for long-range distance calculations and is specifically designed for spherical coordinate systems, making it more suitable than Euclidean distance for most geolocation and trajectory analysis scenarios.

By combining the grid-based representation with Haversine distance calculations, we effectively manage the complexities of geographic data. This approach maintains spatial accuracy and ensures computational efficiency, ultimately enhancing the performance and reliability of our trajectory classification system. Finally, the grid index of each cell is passed as input to the embedding layer.

For temporal representation, the timestamp data is usually a feature composed of day, hour, month, year, minute, and second. DeepeST extracts the hour and day of the week for each timestamp attribute. For instance, DeepeST transformed the timestamp 2009-05-25 20:56:10 into two attributes: the hour of day equal to 20 and the day of the week equal to 0. Finally, the sequence of temporal features is one-hot encoded to input in its corresponding embedding layer.

3.3.2 Subtrajectory embedding

The second challenge of the trajectory classification problem is data sparsity. DeepeST possesses embeddings to receive sequences from the subtrajectories to alleviate the curse of dimensionality, increase privacy, and learn complex mobility data. Embedding is a relatively low-dimensional space into which they can translate high-dimensional vectors. There are several reasons we use subtrajectories embedding:

1. **Dimensionality Reduction**: Traditional methods, such as one-hot encoding, are binary, usually sparse (mostly made of zeros), and very high-dimensional (same dimensionality as the number of distinct labels) (CHOLLET, 2018). Embeddings can significantly reduce the multidimensional nature of check-ins, which include spatial, temporal, and possibly activity or location category information, making them computationally easier to handle while preserving key features. Suppose a student user has visited the POI sequence [Home, Bus Station, University] while another user has visited the POI sequence [Home, Subway, University]. The embeddings of Bus Station and Subway will be similar because they happened in the same context (after Home and before University).
2. **Semantic Capturing**: Embeddings can capture spatial and temporal semantics, such as the relationship between locations and times. This allows deep learning models to

understand better mobility patterns, like frequent routes or preferred places.

3. **Power-law distribution**: The frequency of location in subtrajectories can follow a power-law distribution. Likewise, word embedding in natural language (MIKOLOV *et al.*, 2013), where some POIs of the check-ins are extremely common while others are rare.
4. **Enhanced Comparison and Clustering**: Embeddings facilitate measuring similarities between trajectories, enabling efficient clustering or recommendation algorithms. Similar trajectories result in close embeddings in the vector space, making identifying behavioral patterns or groups easier.
5. **Integration with Neural Networks**: Embeddings are especially beneficial for feeding into neural networks, crucial for deep learning applications. They can be seamlessly incorporated as layers in neural network architectures to model complex relationships in the data.
6. **Data Anonymization**: Representing trajectories as embeddings can also aid in data anonymization, as the resulting vectors are not directly interpretable regarding original locations. This can address privacy concerns without compromising the utility of the data for analysis and modeling.

Deep learning models such as TULER and TULVAE (GAO *et al.*, 2017; ZHOU *et al.*, 2018) explore only the spatial dimension of the embedding vector. DeepeST explores the spatial and temporal dimensions and other features linked to subtrajectories, as illustrated in Figure 8. The more essential features we link to a subtrajectory, the more information to improve classification accuracy and can be used directly and hence save more time.

In summary, DeepeST supports check-in trajectory embeddings from trajectories with multiple dimensions for efficient processing, analyzing, privacy, and applying machine learning techniques to complex mobility data, enhancing the effectiveness and applicability of various analytical tasks.

3.3.3 Recurrent Neural Network - LSTM and BSLTM

DeepeST uses a recurrent layer that receives input from a feature vector but presents embedding layers to each temporal and spatial attribute. So, a concatenation layer is defined between the embedding layers and the recurrent layer to join two embedding vectors in unique input features used in the recurrent layer, as shown in Figure 8.

Recurrent Neural Networks (RNNs) are powerful tools for learning complex patterns

in sequential data, which is not possible with traditional feedforward neural networks. DeepeST uses an LSTM model, which is particularly effective for processing sequences of varying lengths and capturing long-term dependencies in the data. We also experimented with DeepeST using the Bi-directional LSTM (BLSTM) model (SCHUSTER; PALIWAL, 1997), which considers context from both directions of a subtrajectory, eliminating the problem of limited context in feedforward models. LSTM and BLSTM operate at the location, time, and attribute embedding levels to learn the underlying pattern (or label) from the subtrajectory data. Let each subtrajectory $S_i = [(l_1,t_1,A_1),\ldots,(l_n,t_n,A_n)]$, and let h_{e-1}, h_e, and $\tilde{h}_e$ be the last, current, and candidate spatial-time embedding state, respectively. In this context, each point in the subtrajectory, denoted as (l_i,t_i,A_i), includes not just the location l_i and time t_i, but also a set of additional attributes A_i. These attributes might include factors like speed, direction, or environmental conditions, which provide important contextual information. By including A_i in the model, we can give the LSTM more comprehensive data, enabling it to learn and capture more complex patterns within the trajectory, as shown in Eq. 3.1

$$\begin{aligned} i_e &= \sigma(W_i\mathbf{v}^e(l_i,t_i,A_i) + U_i h_{e-1} + V_i c_{e-1} + b_i) \\ f_e &= \sigma(W_f\mathbf{v}^e(l_i,t_i,A_i) + U_f h_{e-1} + V_f c_{e-1} + b_f) \\ o_e &= \sigma(W_o\mathbf{v}^e(l_i,t_i,A_i) + U_o h_{e-1} + V_o c_{e-1} + b_o) \end{aligned} \tag{3.1}$$

These gates in an LSTM function like a series of filters that control the flow of information: the input gate i_e controls how much new information from the current input is added to the cell state c_e, the forget gate f_e determines how much of the previous cell state c_{e-1} should be retained or forgotten, and the output gate o_e decides which parts of the cell state c_e will be passed on as the hidden state h_e. The bias vector b_* represents the bias term for each gate, with b_i, b_f, and b_o being the bias vectors associated with i_e, f_e, and o_e, respectively. σ is a logistic sigmoid function, and matrices W, U, and V ($\in \mathbb{R}^{d\times d}$) are the different gate parameters. $\mathbf{v}^e(l_i,t_i,A_i)$ is the concatenated embedding of the location l_i, time t_i, and other features associated with the point of subtrajectory S_i. The memory cell c_e is updated by partially replacing the existing memory unit with a new cell c_e as illustrated in equation 3.2.

$$c_e = f_e c_{e-1} + i_e \tanh(W_c\mathbf{v}^e(l_i,t_i,A_i) + U_c h_{e-1} + b_c) \tag{3.2}$$

The subtrajectory embedding is updated using equation 3.3, where $\sigma(.)$ denotes the sigmoid function, and $\tanh(.)$ represents the hyperbolic tangent function. The symbol $\odot$ refers to

the element-wise (Hadamard) product. These functions are fundamental in controlling the flow of information through the network, with the sigmoid function squashing inputs to the range [0, 1], and the hyperbolic tangent function squashing inputs to the range [-1, 1].

The final hidden state h_e of the subtrajectory embedding is then calculated using the element-wise product of the output gate o_e and the hyperbolic tangent of the current cell state c_e as shown in equation 3.3.

$$h_e = o_e \odot \tanh(c_e) \tag{3.3}$$

The output of the recurrent neural network denoted as $\tilde{y}$, is passed through a softmax function, which converts the raw output of the network into a probability distribution over the possible labels. The softmax function takes a vector of real numbers and normalizes it, producing a probability for each label. This ensures that the output probabilities sum to 1, making them interpretable as the model's confidence in each label. The resulting probabilities lie in the interval [0, 1].

After applying the softmax function, the probability associated with each label $y \in Y$ is given by equation 3.4, where $k = \{W, U, V, b\}$ represents the set of parameters to be learned by the model. Here, W_{y^i} and b_{y^i} are the weight matrix and bias vector corresponding to label y^i, and $\mathbf{v}(l_i, t_i, A_i)$ is the concatenated embedding of the location, time, and associated attributes for the subtrajectory.

$$\tilde{y^i} = softmax(W_{y^i} h_{y^i} + b_{y^i}) = \frac{\exp\{\mathbf{v}(l_i, t_i, A_i) \cdot S \cdot k_i\}}{\sum_{j=1}^{|Y|} \exp\{\mathbf{v}(l_i, t_i, A_i) \cdot S \cdot k_j\}} \tag{3.4}$$

3.3.4 Optimization in DeepeST

Overfitting is a major problem in RNN due to many weights and biases. To alleviate overfitting, we determined a dropout layer for regularization. Dropout is a strategy radically different from other approaches since it changes the network structure instead of the cost function. Suppose we have a training set X and the corresponding desired output y. Normally, we train by direct propagation of X across the network, and then the backpropagation algorithm computes the error to the gradient. When we use a layer dropout, this process is modified. We randomly (and temporarily) eliminate some of the neurons hidden in the network but leave the input and

output neurons untouched. Heuristically, therefore, dropout can reduce overfitting, whereas other networks adapt differently.

3.3.5 Imbalanced Data in Deep Learning Models

To the best of our knowledge, DeepeST, our proposal, is the first deep-learning model for trajectory classification (TUL Problem) that provides resources to deal with imbalanced classes. An imbalanced classification problem is when the distribution of examples across the known classes is biased or skewed. The distribution can vary from a slight bias to a severe imbalance. In other words, the number of samples in the minority class is smaller than that of the majority class, considering the proportion. For example, twice large or more (FERNÁNDEZ *et al.*, 2008). To address and manage this issue, it is essential to quantify the degree of imbalance within a dataset. Several metrics can be employed to measure and understand this imbalance, providing insights that can guide data preprocessing and model training strategies. Among these metrics, the Imbalance Ratio (IR) and the Coefficient of Variation (CV) stand out for their effectiveness and ease of interpretation.

The Imbalance Ratio (IR) is a vital metric for evaluating the degree of imbalance in a dataset, particularly in classification problems. The IR quantifies the degree of imbalance between the minority class and the majority class. It is defined as the ratio of the number of examples in the majority class (N_{maj}) to the number of examples in the minority class (N_{min}), as presented in Eq. 3.5:

$$IR = \frac{N_{\text{maj}}}{N_{\text{min}}} \tag{3.5}$$

Where:

- N_{maj} is the number of examples in the majority class.
- N_{min} is the number of examples in the minority class.

A higher Imbalance Ratio signifies a greater imbalance, which can challenge the model's ability to accurately learn and predict the minority class. An IR close to 1 indicates that the dataset is well-balanced, with each class having a similar number of instances. As the IR increases, it signifies a growing disparity between the majority and minority classes. A low IR (typically between 1.5 and 3.0) suggests that the imbalance is low, with the majority classes having only slightly more instances than the minority ones (FERNÁNDEZ *et al.*, 2008). A

moderate IR (between 3 and 9) indicates a noticeable imbalance, where the majority classes have significantly more instances than the minority classes, potentially impacting the performance of models trained on this data (FERNÁNDEZ *et al.*, 2008). A high IR (above 9) signifies severe imbalance, with major classes vastly outnumbering the minor ones, which can lead to biased models that perform poorly on underrepresented classes (FERNÁNDEZ *et al.*, 2008). Understanding and measuring the IR is crucial for identifying and addressing class imbalances, ensuring more robust and fair model training.

The Coefficient of Variation (CV) is another metric used to measure imbalance, especially useful in multi-class scenarios. The coefficient of variation represents the ratio of the standard deviation to the mean, and it is a useful statistic for comparing the degree of variation from one data series to another (ABDI, 2010). CV is calculated as the standard deviation (σ) of the number of examples in the classes divided by the mean (μ) number of examples across the classes, as presented in Eq. 3.6.

$$CV = \frac{\sigma}{\mu} \tag{3.6}$$

Where:

- σ is the standard deviation of the number of examples across classes.
- μ is the mean of the number of examples across classes.

The CV provides a relative measure of dispersion, allowing for the comparison of class imbalance across different datasets. A higher CV indicates a greater degree of variability in class distribution, reflecting a more significant imbalance. A CV close to 0% indicates that the class frequencies are very homogeneous, implying minimal variation relative to the mean and suggesting that the classes are well-balanced. A high CV indicates high relative variability, with large differences in class frequencies, suggesting marked imbalance where major classes have many more instances than minor ones (ABDI, 2010).

Most classifiers are built on the assumption that data between classes is balanced and evenly distributed. However, data imbalances can cause major errors and negative repercussions in data analysis, especially in classification jobs. This happens because the algorithms may not learn the patterns of minority classes well. Two solutions for addressing imbalanced datasets are re-sampling and cost-sensitive re-weighting (HAIXIANG *et al.*, 2017).

In re-sampling, considering the data itself perspective using three approaches:

1. over-sampling: we increase the number of samples in the minority class to balance the classes (main techniques are SMOTE and randomly duplicating the minority samples);
2. under-sampling: we eliminate samples from the majority class (an effective method is random undersampling);
3. hybrid approaches: combining the methods of oversampling and under-sampling.

Other approaches are based on cost-sensitive re-weighting; in other words, we influence the loss function by assigning relatively higher costs to examples from minor classes and relatively smaller costs to majority classes. In the literature, cost-sensitive re-weighting has been used for deep learning models (YAN *et al.*, 2016), (LIN *et al.*, 2017), (CUI *et al.*, 2019), (WANG *et al.*, 2016).

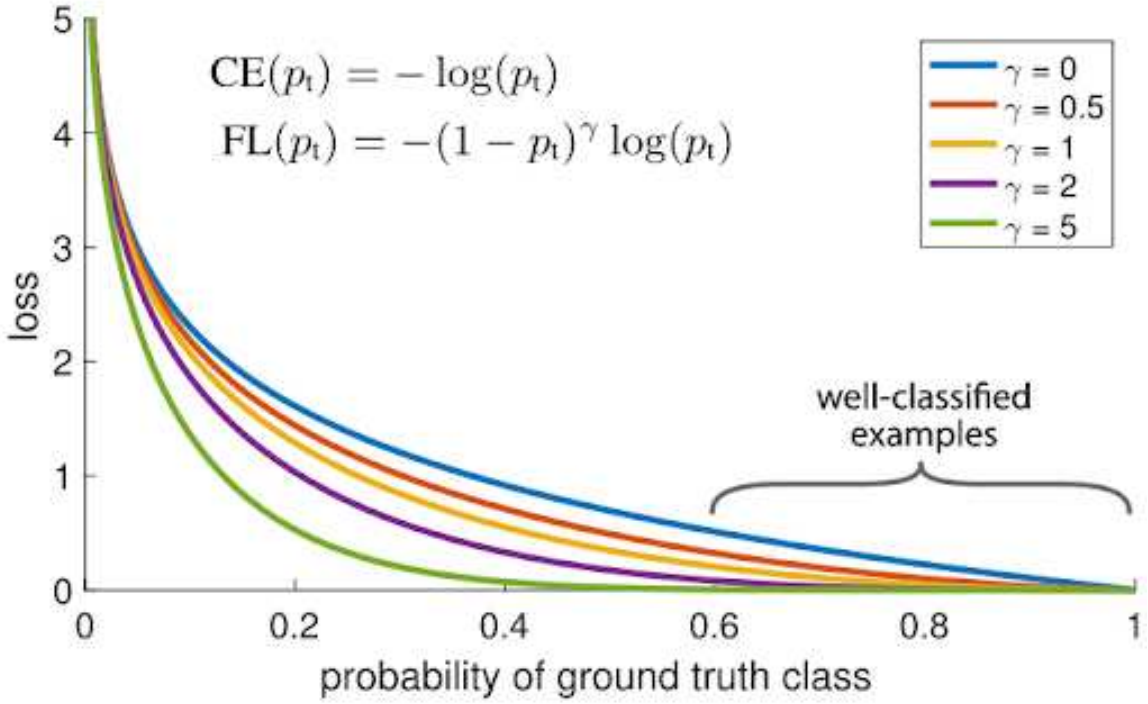
In the context of deep learning using recurrent neural networks like LSTM for trajectory classification, re-sampling approaches can introduce large amounts of duplicated samples, slowing down training and increasing the risk of overfitting. Trajectory data is inherently multi-dimensional, comprising sequences of spatial (latitude, longitude) and temporal (timestamps) points. Techniques like SMOTE (CHAWLA *et al.*, 2002), which are effective for many machine learning problems, are unsuitable here due to the complexity and sequential nature of trajectories, potential for overfitting, loss of real-world variability, and high computational cost. Additionally, under-sampling can discard valuable data, further limiting the model's performance. Therefore, we focus on cost-sensitive re-weighting methods to address class imbalance while preserving the integrity of the trajectory data.

There are two main loss functions for classification tasks using deep learning models: Focal Loss and Class-Balanced Loss.

Focal Loss introduces a modulating term to the traditional cross-entropy loss to focus training on hard negative examples, which are often overlooked in models trained on imbalanced datasets. This adjustment helps to prevent the overwhelming influence of easy examples during the training process. Focal Loss is particularly useful in scenarios where there is an extreme imbalance in the class distribution, as it allows the model to focus more on difficult, misclassified cases that are crucial for robust performance (LIN *et al.*, 2017). It introduces a focusing parameter γ and a balancing factor α, which are used to adjust the contribution of each example to the loss based on the prediction error, as shown in Figure 10.

The modulating factor $(1-p_t)^\gamma$ reduces the loss contribution from easy examples (where p_t is close to 1), amplifying the importance of correcting misclassified examples (LIN *et*

Figure 10 – Focalloss



Source: Lin *et al.* (2017)

al., 2017). The balancing factor α adds a weighting to each class, aiming to balance the dataset's class distribution influence during training. The formal definition of Focal Loss is given by:

$$\mathbf{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{3.7}$$

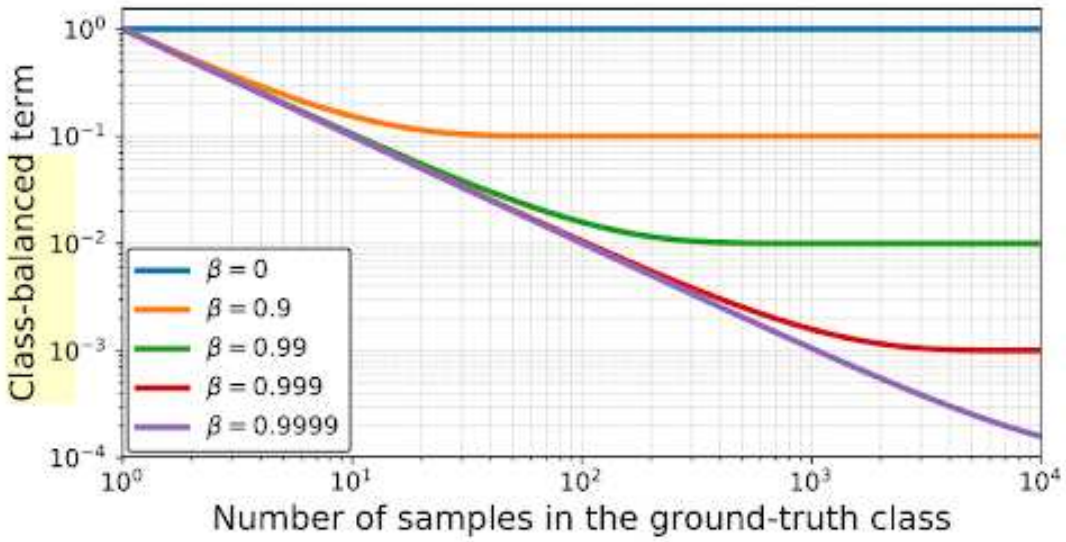
where α_t represents the weighting factor for the class t and γ adjusts the rate at which easy examples are down-weighted, allowing the model to focus more on challenging examples.

Class-Balanced Loss (CBCE) addresses the issue of class imbalance by defining weights inversely proportional to the effective number of samples for each class (CUI *et al.*, 2019), as shown in Figure 11.

This method ensures that minority classes have a higher impact on the loss calculation, compensating for their fewer occurrences in the training data. The effective number of samples for a class i is computed as follows, taking into account a hyperparameter β, which controls the steepness of the weighting function:

$$\mathbf{CBCE}(p_t) = -\frac{1}{E_n}\log(p_t) \tag{3.8}$$

Figure 11 – CBCE



Source: Cui *et al.* (2019)

$$E_n = \frac{1-\beta^{n_i}}{1-\beta} \tag{3.9}$$

The Class-Balanced Loss is particularly effective when combined with deep learning models, as it allows these models to learn more balanced features across classes, leading to improved generalization on less frequent classes.

3.3.6 Loss Functions

DeepeST integrates three loss functions into its training process: Categorical Cross-Entropy (CCE), Class-Balanced Cross-Entropy Loss (CBCE), and Focal Loss (FL). CCE is the standard loss function for multi-class classification tasks where the labels are provided in a one-hot representation. CCE calculates the difference between two probability distributions for the predicted and actual labels across all classes, generally providing a single weighting factor for each class, which may not be effective for imbalanced datasets.

On the other hand, both FL and CBCE are designed to mitigate issues arising from imbalanced training data. FL adjusts the focus of training dynamically, prioritizing hard-to-classify examples (LIN *et al.*, 2017), while CBCE adjusts the weight of each class inversely to its frequency, ensuring that all classes contribute equally to the model's learning process, irrespective of their prevalence in the data (CUI *et al.*, 2019). This comprehensive approach allows DeepeST to achieve more robust and balanced performance across diverse datasets.

4 EXPERIMENTS

In this chapter, we comprehensively evaluate our proposed methods through three distinct experiments conducted on five different datasets to assess DeepeST performance, robustness, and generalizability across varying conditions and data characteristics. Each experiment is structured to highlight specific aspects of our methodology, providing a thorough analysis of its effectiveness and applicability in real-world scenarios. The following sections detail the datasets and results of each experiment.

4.1 Datasets

To evaluate the performance of DeepeST for the classification trajectory problem, we conduct our experiments on three datasets:

1. **Brighkite**[1]: is a location-based social networking service where users share their locations by checking in. The friendship network, collected using their public API, consists of 4,491,143 check-ins from these users over the period from April 2008 to October 2010;
2. **Gowalla**[2]: is a location-based social networking website where users share their locations by checking in. The friendship network was collected using their public API from February 2009 to October 2010, resulting in 6,442,890 check-ins from these users.
3. **Electronic Monitoring Trajectory**: a private criminal dataset extracted from GPS services contains trajectories of offenders in June 2019 who moved around Fortaleza, Ceara.
4. **Foursquare**[3]: is a location-based social networking service that allows users to check in at various locations using a mobile app or website. Users can share their locations with friends, discover new places, and write reviews or tips about the places they visit. The platform uses the check-in data to provide personalized recommendations for restaurants, bars, and other venues based on users' preferences and habits. We collected a public dataset of Foursquare in New York City (NYC), USA, between April 2012 and February 2013. Each check-in is associated with its time stamp, its GPS coordinates, and its semantic meaning.
5. **Weeplaces**[4]: We collected data from Weeplaces, a website designed to visualize users'

[1] https://snap.stanford.edu/data/loc-Brightkite.html
[2] https://snap.stanford.edu/data/loc-gowalla.html
[3] https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[4] https://www.yongliu.org/datasets/

check-in activities on location-based social networks (LBSNs). Weeplaces integrates with the APIs of several LBSN services, including Facebook Places, Foursquare, and Gowalla. Users can log into Weeplaces using their LBSN accounts and connect with friends who also use this application. Foursquare generated initially all the data in this dataset. It comprises 7,658,368 check-ins made by 15,799 users across 971,309 locations. Due to data collection limitations, we are unable to retrieve the original Foursquare IDs of the Weeplaces users. However, we can access their check-in histories, friends who also use Weeplaces, and additional location information.

4.2 Experiment 01: Machine Learning and Deep Learning for Trajectory Classification

In this section, we present the first experimental evaluation to evaluate DeepeST in terms of quality prediction. For reproducibility purposes, we made the source code available on GitHub [5]. We start by providing details about the data preparation, the baseline algorithms, and the evaluation metrics, followed by the experimental evaluation. We used Gowalla, Brightkite, and Electronic Monitoring Trajectory Datasets in this experiment. Our first case study tackles the main objectives:

1. Assess individually the DeepeST models with state-of-the-art machine learning and deep learning models for the trajectory user-linking from check-ins (Gowalla and Brightkite).
2. Assess individually the DeepeST models with machine learning approaches for criminal activity detection from GPS-based trajectories.
3. Assess the performance between LSTM and BILSTM.

4.2.1 Data Preparation

The classification task for Brightkite and Gowalla datasets is to predict the corresponding user who generated a given sub-trajectory. We decided to use temporal segmentation instead of other approaches, such as distance-based, speed-based, or stop-point-based segmentation (ZHENG, 2015), due to the nature of the datasets, which consists of check-in data from Location-Based Social Networks (LBSNs) containing User, Hour, timestamp, and poi-category, as shown in Table 2.

Temporal segmentation is particularly suitable for LBSN check-in data for several reasons.

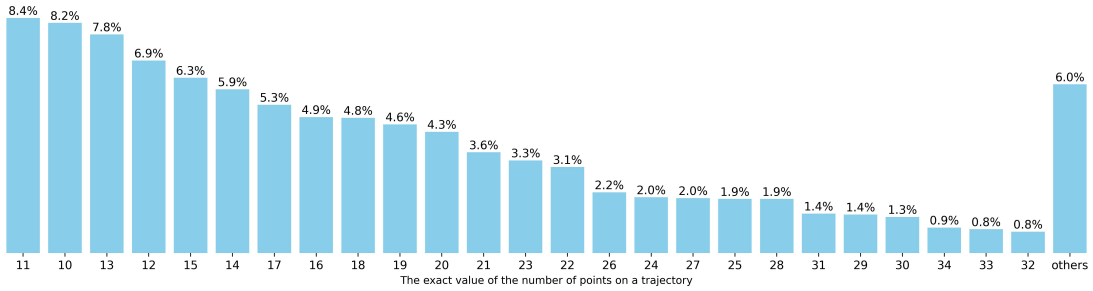
[5] https://github.com/nickssonarrais/ICAART2021

1. First, LBSN check-ins are inherently timestamped events, making temporal information a primary and reliable dimension for segmentation. This allows for a natural partitioning of the data based on the timing of user activities, reflecting their real-world behavior patterns more accurately.
2. Second, temporal segmentation aligns well with the typical usage patterns of LBSNs, where users check in at various POIs at specific times, such as during their commute, lunch breaks, or leisure activities. By segmenting the data temporally, we can capture these daily or weekly routines, which are crucial for accurately linking trajectories to users.
3. Temporal segmentation avoids the potential pitfalls of distance-based or speed-based methods, which may be less effective in urban environments with dense POIs and frequent short-distance check-ins. Distance-based methods cluster nearby check-ins at significantly different times, leading to inaccurate trajectory representations. Similarly, speed-based segmentation could misinterpret slow movement within a crowded area as stop points, missing the actual flow of user activities.
4. Stop point-based segmentation might not be suitable for LBSNs because users often check in without necessarily stopping for extended periods. Instead, they may perform quick check-ins as they pass by or briefly visit various locations.

Gao *et al.* (2017) also uses temporal segmentation but applies daily segmentation to transform sequences of points into sub-trajectories. We did not do daily segmentation because more than 50% of daily trajectories for Brightkite and Gowalla datasets have only one trajectory point, causing low-performance values for the models, as in (GAO *et al.*, 2017). We use a segmentation based on time to create weekly sub-trajectories from each user. Our distribution of the number of trajectories by the number of points is shown in Figures 12 and 13. Note that our distribution becomes greater even when we segment the data by week. Note that our distribution is more uniform considering weekly trajectories.

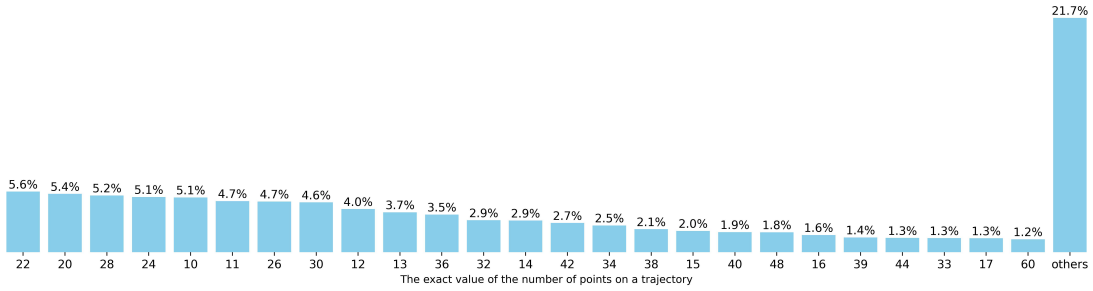
We defined a grid covering all spatial points of trajectories with a cell size of $30m^2$. This approach allows us to discretize the continuous space into manageable units, simplifying the representation of geographic locations and making the data more accessible to handle and analyze. We created a grid to cover all data points in the dataset, with a small margin to ensure coverage of all relevant areas. This focused grid setup enhances computational efficiency and accuracy, ensuring our trajectory classification system can effectively process and analyze the geospatial data.

Figure 12 – Percentage of trajectories by the number of points in the weekly segmentation for Gowalla



Source: Created by the author

Figure 13 – Percentage of trajectories by the number of points in the weekly segmentation for Brightkite



Source: Created by the author

We selected only sub-trajectories of users with at least 15 weekly trajectories because we will have at least two samples for each user in the validation and test sets. Finally, the Brightkite dataset contains 4565 sub-trajectory samples in the train set, 996 in the validation set, and 1085 in the test set. The Gowalla dataset contains 2325 sub-trajectories sequences in the train set, 517 in the validation set, and 572 in the test set. Table 2 describes the attributes of trajectory points to both Brightkite and Gowalla datasets.

Table 2 – Description of the check-in trajectories to Brightkite and Gowalla

Attributes	**Type**	**Range/example**	**N.**
User	Nominal	{58186, ..., 58190}	{197, 100}
Weekday	Nominal	{Monday, ..., Sunday}	7
Hour	Numeric	[0, 23]	24
Index grid	Numeric	{0, ..., 46458}	{3742, 11345}
POI	Nominal	{dsda411, ..., ee8b8e}	{4085,15977}
Subtraj ID	Numeric	{0, ..., 10000}	{6646, 2335}

Source: Created by the author

The classification task is to identify criminal activities and what the criminal is doing

- at home, blocking the signal from the equipment that transmits their location, selling stolen car parts, in a hearing with the judge, among other categories. In this dataset, we randomly selected the trajectories of ninety offenders. For the criminal activity dataset, we used a different segmentation based on time and label to avoid overlapping sub-trajectories. We created sub-trajectories to group each user and criminal activity for up to thirty minutes. In order words, for each offender, the algorithm returns sub-trajectories containing a single criminal activity for up to thirty minutes. We define a 2D grid content cells of $30m^2$ for an area around Fortaleza city. Table 3 describes the attributes of trajectory points to the Criminal dataset. Finally, the Criminal dataset contains 116,255 sub-trajectory sequences in the train set, 24,912 in the validation set, and 24,912 in the test set.

Table 3 – Description of the GPS trajectories to the Criminal Dataset

Attributes	**Type**	**Range/example**	**N.**
Offender	Nominal	{58186, ..., 58190}	90
Weekday	Nominal	{Monday, ..., Sunday}	7
Hour	Numeric	[0, 23]	24
Index grid	Numeric	{0, ..., 46458}	36690
Criminal Activity	Nominal	{home, ..., blocked signal}	9
Subtraj ID	Numeric	{0, ..., 10000}	166079

Source: Created by the author

To validate the models, we split the three datasets into training (70%), validation (15%), and test sets (15%). To eliminate the risk of overfitting, the test set was kept completely separate from the training set. Since we are working with trajectory data, we used a temporal window to divide the data for each user, ensuring that the sequence of events was preserved. The baseline algorithms were then run ten times using the training and validation sets, and the final evaluation was performed on the test set.

We evaluate the models using Accuracy, Macro Precision, Macro Recall, and Macro F1-Score. We specifically used Macro metrics because our dataset is imbalanced, and Macro averaging treats all classes equally, providing a more comprehensive evaluation of model performance across all classes. This approach ensures robust validation and reliable performance metrics.

We apply the grid-search technique to combine several hyperparameters to find the optimal set for each model. For the DeepeST models (the one that uses LSTM, the one with BILSTM), BITULER, and TULVAE, we keep 64 as the batch size and 0,001 as the learning rate and vary the units (**un**) of the recurrent layer, the embedding size to each attribute (**es**) and the

dropout (**dp**). For TULVAE, we also vary the latent variable (**z**). We determine an early stopping callback, that is, a stop training when, in our case, the accuracy has stopped improving. We set the early stopping as 20 for the patience argument to minimize overfitting, i.e., the number of epochs that produced the model's accuracy with no improvement, after which training should be stopped. For further details, refer to Keras library [6].

4.2.2 Baselines algorithms

We compare DeepeST with four state-of-the-art approaches from the field of machine learning and deep learning classification: XGBoost (CHEN; GUESTRIN, 2016), Random Forest (BREIMAN, 2001), BITULER (GAO *et al.*, 2017), and TULVAE (ZHOU *et al.*, 2018). Detailed information about each method can be found in subsections 2.3.1 and 2.3.2.

For the XGBoost model, we vary the number of estimators (**ne**), the maximum depth of a tree (**md**), the learning rate (**lr**), the gamma (**gm**), the fraction of observations to be random samples for each tree (**ss**), the sub-sample ratio of columns when constructing each tree (cst), the regularization parameters (**l1**) and (**l2**). We also set the early stopping round to 20 for XGBoost.

For Random Forest, we vary the number of trees (**ne**), the maximum number of features to consider at every split (**mf**), the maximum number of levels in a tree (**md**), the minimum number of samples required to split a node (**mss**), the minimum number of samples needed for each leaf node (**msl**), and finally, the method of selecting samples for training each tree (**bs**).

For BITULER and TULVAE, we vary the units (**un**), the embedding size (**es**) of the POI identifier, and the dropout (**dp**). We also set the early stopping round to 20, the learning rate to 0,001, and the batch size to 64 (the same settings used in the DeepeST). For more details about parameters, we refer to Git Hub repository [7].

For what concerns DeepeST variations, a sub-trajectory S is a sequence with each of the following attributes $(ig_i, hr_i, poi_i, wk_i)$, where ig is the index grid cell and poi is the POI identifier, wk is the weekday, and hr is the hour); BITULER and TULVAE only deal with one feature, so the input is a sequence of POI identifier as presented in (GAO *et al.*, 2017; ZHOU *et al.*, 2019). For XGBoost and RandomForest, a sub-trajectory is a unique concatenated sequence $[ig_1, ig_2, ..., wk_{n-1}, wk_n]$ with all attributes $(ig_i, hr_i, poi_i, wk_i)$.

[6] https://keras.io/

[7] https://github.com/nickssonarrais/ICAART2021

There are two variations for DeepeST concerning the network layers: one with LSTM (DeepeST-LSTM) and another with BILSTM (DeepeST-BILSTM).

4.2.3 Performance Comparison

From the results reported in Tables 4 and 5, we summarize the performance comparison between the variants of DeepeST, XGBoost, RandomForest, BITULER, and TULVAE for the Brightkite and Gowalla datasets. The two best values are highlighted in **bold**, and the third is shown as *underlined*. We can see that the DeepeST overcame the baselines. In summary, in comparison with machine learning approaches, we can notice that DeepeST outperforms XGBoost and Random Forest across all metrics by up to 11% in Brightkite and by up to 31% in Gowalla, considering F1-Score. DeepeST takes advantage of the LSTM/BILSTM and operates at embedding levels to learn the underlying user categories from check-in sub-trajectory data. RNN models (LSTM/BILSTM) are proper models to learn from temporal sequences as sub-trajectories.

We can notice that DeepeST also outperforms BITULER and TULVAE across all metrics by up to 5% in Brightkite and by up to 7%, considering the F1-score. DeepeST built a more robust model using a set of variables instead of only using the points of interest identification. As we can see, only POI identification can't distinguish different users. It is worth noting that the results could be higher if there were more critical features to separate the classes from many users in the dataset (maybe features based on external events and weather conditions). The expert's view of the application domain can be essential to increase the model's performance. We applied only the spatial and time features (weekday, hour, index grid, and POI) extracted from the original Gowalla and Brightkite datasets.

DeepeST-LSTM only preserves past information because the inputs it has seen are from the past. DeepeST-BILSTM runs your inputs in two ways, one from past to future and one from future to past. In our experiments, DeepeST-BILSTM achieved slightly more significant results than DeepeST-LSTM for the Brightkite dataset since DeepeST-BILSTM takes into account an effectively infinite amount of context on both sides of a sub-trajectory position and eliminates the problem of limited context that applies to any feed-forward model. On the other hand, DeepeST-LSTM achieved slightly more significant results than DeepeST-BILSTM for the Gowalla dataset. It is essential to highlight that the results between DeepeST-BILSTM and DeepeST-LSTM are very close. However, using future information can usually be easier and

faster for the network to understand the next label.

Table 4 – Results to Brightkite

Method	Best set of the grid search	Accuracy		Precision		Recall		F1-Score	
		mean	std	mean	std	mean	std	mean	std
DeepeST-LSTM	un:400, es:100, dp:0.5	**0,9591**	0,0018	**0,9644**	0,0048	**0,9514**	0,0025	**0,9512**	0,0033
DeepeST-BILSTM	un:100, es:100, dp:0.5	**0,9632**	0,0030	**0,9658**	0,0038	**0,9563**	0,0044	**0,9557**	0,0041
BiTULER	un:100, es:100, dp:0.5	0,9372	0,0052	0,9417	0,0055	0,9211	0,0068	0,9234	0,0066
TULVAE	un:100, es:300, dp:0.5, z:300	0,9452	0,0044	0,9439	0,0062	0,9325	0,0048	0,9308	0,0054
Random Forest	ne:200, md:30, mss:5, msl:1, mf:auto, bs:False	0,8717	0,0043	0,8744	0,0090	0,8431	0,0060	0,8440	0,0072
XGBoost	ne:2000, md:5, gm:0, ss:0.8, cst:0.5, l1:1, l2:100	0,8769	0,0047	0,8717	0,0059	0,8481	0,0063	0,8483	0,0065

Table 5 – Results to Gowalla

Method	Best set of the grid search	Accuracy		Precision		Recall		F1-Score	
		mean	std	mean	std	mean	std	mean	std
DeepeST-LSTM	un:100, es:400, dp:0.5	**0,9760**	0,0039	**0,9821**	0,0027	**0,9744**	0,0042	**0,9750**	0,0039
DeepeST-BILSTM	un:200, es:100, dp:0.5	**0,9739**	0,0038	**0,9798**	0,0034	**0,9727**	0,0040	**0,9723**	0,0044
BiTULER	un:300, es:400, dp:0.5	0,9122	0,0050	0,9274	0,0070	0,9101	0,0060	0,9078	0,0072
TULVAE	un:100, es:300, dp:0.5, z:300	0,9159	0,0085	0,9338	0,0121	0,9111	0,0096	0,9105	0,0109
Random Forest	ne:400, md:30, mss:2, msl:2, mf:sqrt, bs:False	0,7047	0,0088	0,7020	0,0139	0,6841	0,0093	0,6631	0,0109
XGBoost	ne:2000, md:10, gm:0, ss:0.8, cst:0.5, l1:1, l2:100	0,6545	0,0112	0,6393	0,0197	0,6327	0,0134	0,6143	0,0152

Table 6 shows the best set of parameters from the grid search, and also summarizes the performance comparison between the variants of DeepeST, XGBoost, and Random Forest. It is essential to mention that BITULER and TULVAE were not included in the experiments since they are applied from a sequence of POI identifiers (one-dimensional data) in check-in trajectories. To DeepeST variations, a sub-trajectory S is a sequence for each attribute in (ig_i, hr_i, wk_i), where ig is the Index grid, hr is the hour of day, and wk is the weekday. For XGBoost and Random Forest, a sub-trajectory is a unique concatenate sequence $[ig_1, ig_2, ..., wk_{n-1}, wk_n]$ with all attributes (ig_i, hr_i, wk_i). The two best values are highlighted in **bold**, and the third is shown as *underlined*. We can notice that DeepeST outperforms XGBoost and Random Forest on all metrics by up to 25%, considering F1-macro. DeepeST again takes advantage of the LSTM/BILSTM and operates at the location and time embedding levels to learn the underlying criminal categories from the offenders' sub-trajectory data. DeepeST-BILSTM achieved slightly more significant results than DeepeST-BILSTM for the Criminal dataset.

Table 6 – Results to Criminal Dataset

Method	Best set of the grid search	Accuracy		Precision		Recall		F1-Score	
		mean	std	mean	std	mean	std	mean	std
DeepeST-LSTM	un:100, es:400, dp:0.5	**0,9188**	0,0010	**0,8792**	0,0075	**0,8283**	0,0043	**0,8504**	0,0040
DeepeST-BILSTM	un:200, es:100, dp:0.5	**0,9203**	0,0013	**0,8826**	0,0071	**0,8365**	0,0052	**0,8564**	0,0026
Random Forest	ne:400, md:30, mss:2, msl:2, mf:sqrt, bs:False	0,7917	0,0002	0,6806	0,0017	0,5515	0,0010	0,5910	0,0012
XGBoost	ne:2000, md:10, gm:0, ss:0.8, cst:0.5, l1:1, l2:100	0,8121	0,0008	0,6671	0,0017	0,5765	0,0015	0,6084	0,0011

4.3 Experiment 02: Deep Learning for TUL Problem in Imbalanced Datasets

In this section, we delve into the second experiment to evaluate DeepeST in terms of quality prediction. We start by providing details about the data preparation, the baseline algorithms, and the evaluation metrics, followed by the experimental evaluation. In this experiment, we used Gowalla, Brightkite, and Foursquare datasets. Our second case study tackles the main objectives:

1. Assess CCE and CBCE loss function to imbalanced datasets varying LSTM and BILSTM using three real datasets from LBSN.
2. Assess the DeepeST models individually with state-of-the-art (MARC, BI-TULER, TUL-VAE) for Trajectory User Liking from LSBN trajectories.
3. Assess the impact of DeepeST hyperparameters such as Embedding size, Hidden Units, and Cell size of the Grid Index.

4.3.1 Data Preparation

We selected only sub-trajectories of users with at least 15 weekly trajectories because we will have at least two samples for each user in the validation and test sets. Table 7 describes the attributes of trajectory points to Brightkite, Gowalla, and Foursquare NYC datasets.

Table 7 – Description of the check-in trajectories for Brightkite, Gowalla, and Foursquare NYC

Attributes	**Type**	**Range/example**	**N.**
User	Nominal	{58186, ..., 58190}	{197, 147, 81}
Weekday	Nominal	{Monday, ..., Sunday}	7
Hour	Numeric	[0, 23]	24
Index grid	Numeric	{0, ..., 46458}	{3742, 11345, 7754}
POI	Nominal	{dsda411, ..., ee8b8e}	{4085, 15977, 8064}
Subtraj ID	Numeric	{0, ..., 10000}	{6646, 2335, 1749}

Source: Created by the author

We use segmentation based on time and create weekly sub-trajectories from each user check-in, as performed in (FREITAS *et al.*, 2021c; May Petry *et al.*, 2020). We selected only sub-trajectories of users with at least 15 weekly sub-trajectories because we will have at least two samples for each user in the validation and test sets. For the Grid index approach in DeepeST, we created a virtual grid cell and set a cell size of 30m, covering all points for each dataset. Therefore, each latitude and longitude is mapped to a 30m x 30m region. We use 30 meters because we believe it is sufficient to separate even minor points of interest, like bars, homes, and restaurants. For the GeoHash representation used in MARC, we use Base32 in

GeoHash for building instances, as performed in (May Petry *et al.*, 2020).

CBCE is designed to address training from imbalanced data by introducing a weighting factor inversely proportional to the effective number of samples (CUI *et al.*, 2019). We used the training set to define the weights for CBCE, ensuring that the model appropriately accounts for the imbalance during the training process. This approach enhances the model's ability to learn from minority classes, contributing to more balanced and accurate performance across all classes.

We apply the grid-search technique to combine several hyperparameters for each model to find the optimal set of hyperparameters. Although it is extremely computationally expensive and may take a long time to run your machine, grid-search ensures that we find the best hyperparameters considering each model and dataset. We keep 64 batch size and 0,001 to learning rate for all the models, and 0,5 to dropout (**dp**). We also vary the units (**un**) of the recurrent layers, the embedding size to each attribute (**es**), and the units of the dense layer used after the GeoHash attribute (**ds**). We determine a first stopping callback, which is a stop training when, in our case, the accuracy has stopped improving. We set the early stopping as 20 for the patience argument to minimize overfitting, i.e., the number of epochs that produced the model's accuracy with no improvement, after which training should be stopped. For further details, refer to Keras library [8]. Exclusively for TULVAE, besides varying the units (**un**) and the embedding size (**es**) of the POI identifier, we also vary the latent variable (**z**). For more details about parameters or reproducibility purposes, we made the source code available on GitHub [9].

To validate the models, we split the three datasets into training (70%), validation (15%), and test sets (15%). To eliminate the risk of overfitting, the test set was kept completely separate from the training set. Since we are working with trajectory data, we used a temporal window to divide the data for each user, ensuring that the sequence of events was preserved. The baseline algorithms were then run ten times using the training and validation sets, and the final evaluation was performed on the test set. We compared the models using Accuracy, Macro Precision, Macro Recall, and Macro F1-Score. We specifically used Macro metrics because our dataset is imbalanced, and Macro averaging treats all classes equally, providing a more comprehensive evaluation of model performance across all classes. This approach ensures robust validation and reliable performance metrics.

[8] https://keras.io/

[9] https://github.com/nickssonarrais/DeepeST-FLAIRS

4.3.2 Baselines algorithms

There are four variations for DeepeST concerning the network layers and loss functions: two with BILSTM, one configured with CCE loss (DeepeST-BILSTM-CCE), and another CBCE loss (DeepeST-BILSTM-CBCE). Two with LSTM, one configured with CCE loss (DeepeST-LSTM-CCE), and another with CBCE loss (DeepeST-LSTM-CBCE). We compare DeepeST variation with three state-of-the-art approaches from the field of deep learning classification for Trajectory User-Liking:

1. BiTULER: is a deep learning method that uses LSTM to link trajectory users to their subtrajectories based on POI identity (GAO *et al.*, 2017).
2. TULVAE: a generative model for mining human movement patterns. It learns the underlying hierarchical structures of trajectories and addresses data sparsity through semi-supervised learning (ZHOU *et al.*, 2018).
3. MARC: a deep learning-based method that leverages spatial, temporal, and semantic variables to characterize multiple-aspect trajectories. It employs a multi-attribute embedding layer to encode the diverse dimensions (May Petry *et al.*, 2020).

4.3.3 Performance Comparison

From the results reported in Tables 8, 9 and 10, we summarize the performance comparison between the variants of DeepeST, MARC, BITULER, and TULVAE using all available features for the three datasets. We highlighted the two best values in **bold** and the third one in *underlined*. For what concerns DeepeST variations and MARC, a sub-trajectory S is a sequence with each of the following attributes $(ig_i, hr_i, poi_i, wk_i)$, where *ig* is the grid index cell, and *poi* is the POI identifier, *wk* is the weekday, and *hr* is the hour); BITULER and TULVAE only deal with one feature, so the input is a sequence of POI identifier as presented in (GAO *et al.*, 2017; ZHOU *et al.*, 2019).

From the results reported in Tables 8, 9 and 10, we can summarize the performance comparison between the variants of DeepeST, MARC, BITULER, and TULVAE using all available features for the three datasets. We highlighted the two best values in **bold** and the third in *underlined*. We can see that DeepeST with (LSTM/BILSTM) using all features combined in sub-trajectory classification yields improvements over the baselines on the three datasets, considering accuracy, precision, recall, and F1-score.

For the Brightkite dataset (Table 8), DeepeST-BILSTM-CBCE outperforms all other models in terms of balanced accuracy (0.9757), precision (0.9738), recall (0.9695), and f1-score (0.9687). DeepeST-LSTM-CCE also shows strong performance, with its mean values for these metrics being consistently high, though slightly lower than DeepeST-BILSTM-CBCE. DeepeST-BILSTM-CBCE follows closely in terms of mean values. MARC performs well but falls short compared to DeepeST variants, while BITULER and TULVAE show significantly lower performance across all metrics.

Table 8 – Classification results on stratified holdout evaluation in Brighkite dataset

Model	Bal. Accuracy		Precision		Recall		F1-Score	
	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,9757**	0,0021	**0,9738**	0,0037	**0,9695**	0,0026	**0,9687**	0,0033
DeepeST-BILSTM-CCE	0,9729	0,0017	**0,9716**	0,0034	0,9691	0,0020	**0,9669**	0,0027
DeepeST-LSTM-CBCE	0,9735	0,0017	0,9715	0,0035	**0,9693**	0,0026	0,9668	0,0032
DeepeST-LSTM-CCE	**0,9748**	0,0019	0,9706	0,0043	0,9678	0,0031	0,9660	0,0036
MARC	0,9718	0,0033	0,9685	0,0061	0,9642	0,0043	0,9628	0,0052
BITULER	0,9520	0,0053	0,9470	0,0068	0,9350	0,0057	0,9364	0,0060
TULVAE	0,9581	0,0012	0,9459	0,0028	0,9426	0,0021	0,9403	0,0022

Source: Created by the author

In the Gowalla dataset (Table 9), DeepeST-BILSTM-CBCE demonstrates the best performance, achieving the highest mean values for balanced accuracy (0.9688), precision (0.9608), recall (0.9645), and f1-score (0.9574). DeepeST-LSTM-CBCE closely follows, showing very similar performance. DeepeST-BILSTM-CCE and DeepeST-LSTM-CCE also perform well but are slightly behind in mean values. MARC, BITULER, and TULVAE show considerably lower performance, indicating that DeepeST variants are more effective for this dataset.

Table 9 – Classification results on stratified holdout evaluation in the Gowalla dataset

Model	Bal. Accuracy		Precision		Recall		F1-Score	
	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,9688**	0,0021	**0,9608**	0,0024	**0,9645**	0,0016	**0,9574**	0,0022
DeepeST-BILSTM-CCE	0,9665	0,0044	0,9577	0,0075	0,9619	0,0039	0,9541	0,0058
DeepeST-LSTM-CBCE	**0,9682**	0,0016	**0,9609**	0,0053	**0,9638**	0,0031	**0,9568**	0,0043
DeepeST-LSTM-CCE	0,9669	0,0024	0,9587	0,0062	0,9619	0,0037	0,9541	0,0051
MARC	0,9456	0,0060	0,9358	0,0122	0,9457	0,0063	0,9340	0,0091
BITULER	0,9008	0,0111	0,8964	0,0126	0,8927	0,0117	0,8782	0,0140
TULVAE	0,9199	0,0092	0,9240	0,0143	0,9210	0,0137	0,9110	0,0124

Source: Created by the author

For the Foursquare NYC dataset (Table 10), DeepeST-BILSTM-CBCE again leads across all metrics with the highest balanced accuracy (0.9957), precision (0.9961), recall (0.9948), and f1-score (0.9947). DeepeST-BILSTM-CCE and DeepeST-LSTM-CBCE also show excellent

performance, with slightly lower but still very high values. MARC, BITULER, and TULVAE perform worse than the DeepeST variants, with TULVAE showing the lowest results among all models.

Table 10 – Classification results on stratified holdout evaluation in the Foursquare NYC dataset

Model	Bal. Accuracy		Precision		Recall		F1-Score	
	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,9957**	0,0022	**0,9961**	0,0022	**0,9948**	0,0027	**0,9947**	0,0026
DeepeST-BILSTM-CCE	**0,9933**	0,0016	0,9930	0,0015	**0,9919**	0,0016	**0,9918**	0,0016
DeepeST-LSTM-CBCE	0,9927	0,0038	**0,9931**	0,0030	0,9915	0,0034	0,9915	0,0036
DeepeST-LSTM-CCE	0,9927	0,0021	0,9930	0,0020	0,9914	0,0020	0,9914	0,0021
MARC	0,9893	0,0052	0,9913	0,0044	0,9883	0,0052	0,9878	0,0058
BITULER	0,9890	0,0039	0,9917	0,0042	0,9877	0,0041	0,9880	0,0049
TULVAE	0,9820	0,0050	0,9854	0,0043	0,9809	0,0052	0,9798	0,0058

Source: Created by the author

DeepeST takes advantage of the CBCE/CCE, data representation using Grid Index and embedding, and LSTM/BILSTM, which operates at embedding levels to learn the underlying user categories from check-ins sub-trajectory data. RNN models (LSTM/BILSTM) are proper models to learn from temporal sequences as sub-trajectories. We notice that MARC outperforms BITULER and TULVAE across all metrics in most datasets. MARC and DeepeST built a more robust model using a set of variables instead of only a POI identifier. We can see that POI identification may not distinguish different users in Gowalla and Brightkite. In Foursquare NYC, all models reached values above 98% for accuracy, precision, and recall and above 97% for F1-macro. This occurs when only the spatial feature is relevant to detecting user mobility patterns. It is worth noting that the results of MARC and DeepeST for all datasets could be higher if more relevant features existed to separate the classes from different users in the dataset (maybe features based on external events and features that characterize different people). The expert's view of the application domain can be essential to increase the model's performance.

Comparing DeepeST variations and MARC, note that DeepeST yields improvements over MARC on the three datasets (Gowalla, Foursquare, and Brightkite), considering accuracy, precision, recall, and F1-score. MARC uses GeoHash, connected to a dense layer to represent the spatial feature. Meanwhile, DeepeST uses the Grid Index, linked to an embedding layer. Using the Grid Index linked to an embedding provided better results because an embedding layer approximates the input data to a recurrent neural network. The embedding output is denser than the dense layer output. Grid index generates an integer vector that can be directly provided for an embedding layer, considering that we use a padding sequence to ensure the fixed-size sequence.

Using the GeoHash approach presented in (May Petry *et al.*, 2020), there is a transformation to a binary matrix that cannot provide input for an embedding layer. In this case, the GeoHash uses a dense layer that makes the sparse data to the RNN. When we use a dense layer, we lose the ability to make the data denser, as in the embedding layer.

Regarding DeepeST and its variations, we generally note that DeepeST-BILSTM provided improvements over DeepeST-LSTM. DeepeST-BILSTM runs its inputs in two ways, one from past to future and one from future to past. In our experiments, DeepeST-BILSTM achieved slightly more significant results than DeepeST-LSTM for the three datasets. DeepeST-BILSTM considers an effectively infinite amount of context on both sides of a sub-trajectory position and eliminates the problem of limited context that applies to any feed-forward model. It is essential to highlight that the results between DeepeST-BILSTM and DeepeST-LSTM are very close. However, DeepeST-BILSTM proved to be better and faster at understanding contexts using past and future information.

If we look at the solutions for coping with imbalanced datasets, we can notice that CBCE loss provides improvements in the recall model. Note that DeepeST-BILSTM-CBCE achieved the highest recall values in all the datasets. CBCE quantifies the weighting factor inversely proportional to the effective number of samples per class. In other words, considering the dataset distribution to each user, the CBCE provides a lower weight to the majority classes and a higher weight to the minority classes. In this way, the algorithm learns the patterns of the minority classes. When we use CCE, the weight factor is the same for all classes, so the algorithm does not understand the patterns in minority classes.

Overall, the DeepeST models, especially those using BILSTM with CBCE and CCE loss functions, consistently outperform the baseline models (MARC, BITULER, and TULVAE) across all datasets and metrics. This indicates the effectiveness of the DeepeST approach in handling the complexities of trajectory user-linking in multidimensional and imbalanced datasets.

4.3.4 Performance Evaluation using Top-K Accuracy

Evaluating performance using Top-K accuracy is crucial in multi-class classification tasks, as it provides a more nuanced understanding of a model's predictive capabilities. With such many classes, achieving exact predictions becomes increasingly complex, making it crucial to consider a broader set of potential outcomes. This metric is particularly valuable in applications where identifying the correct class within the top K predictions is essential, such as user identifi-

cation and recommendation systems. Top-K accuracy allows us to assess the model's ability to rank the actual class among its highest probability predictions, offering insights into its practical effectiveness and robustness. This evaluation provides a more comprehensive assessment of the model's performance beyond top-1 accuracy (ACC@1), top-5 accuracy (ACC@5), and top-10 accuracy (ACC@10), highlighting its utility in real-world scenarios.

From the results reported in Tables 11, 12 and 13, we summarize the performance comparison between the variants of DeepeST using all available features for the three datasets. In Table 11, the performance on the Brightkite dataset shows that the DeepeST-BILSTM-CBCE model achieves the highest ACC@1 (0,9612), ACC@5 (0,9863), and ACC@10 (0,9904) among the four models. The standard deviations are relatively low, indicating stable performance. The other models, DeepeST-BILSTM-CCE, DeepeST-LSTM-CBCE, and DeepeST-LSTM-CCE, also perform well but with slightly lower accuracy and similarly low standard deviations.

Table 11 – Evaluation probabilities from Top-K Accuracy for the Brightkite Dataset

Model	**ACC@1**		**ACC@5**		**ACC@10**	
	mean	**std**	**mean**	**std**	**mean**	**std**
DeepeST-BILSTM-CBCE	0,9612	0,0019	0,9863	0,0016	0,9904	0,0015
DeepeST-BILSTM-CCE	0,9604	0,0027	0,9855	0,0020	0,9900	0,0010
DeepeST-LSTM-CBCE	0,9606	0,0023	0,9855	0,0011	0,9896	0,0018
DeepeST-LSTM-CCE	0,9583	0,0025	0,9850	0,0016	0,9887	0,0016

Source: Created by the author

In Table 12, we show the evaluation probabilities for the Gowalla dataset. DeepeST-BILSTM-CBCE again outperforms the other models with an ACC@1 of 0.9705, ACC@5 of 0.9881, and ACC@10 of 0.9902. The performance differences among the models are more pronounced in this dataset, particularly in ACC@1, where DeepeST-BILSTM-CCE and DeepeST-LSTM-CBCE show lower mean values and higher standard deviations.

Table 12 – Evaluation probabilities from Top-K Accuracy for the Gowalla dataset

Model	**ACC@1**		**ACC@5**		**ACC@10**	
	mean	**std**	**mean**	**std**	**mean**	**std**
DeepeST-BILSTM-CBCE	0,9705	0,0025	0,9881	0,0020	0,9902	0,0015
DeepeST-BILSTM-CCE	0,9649	0,0055	0,9848	0,0022	0,9892	0,0018
DeepeST-LSTM-CBCE	0,9677	0,0069	0,9862	0,0048	0,9899	0,0024
DeepeST-LSTM-CCE	0,9678	0,0045	0,9851	0,0019	0,9899	0,0018

Source: Created by the author

In Table 13, which reports the results for the Foursquare NYC dataset, all models

achieve exceptionally high accuracies. DeepeST-BILSTM-CBCE and DeepeST-LSTM-CBCE achieved perfect scores in ACC@5 and ACC@10 (1.0000). The ACC@1 for DeepeST-BILSTM-CBCE is slightly higher (0.9987) compared to the other models, with DeepeST-BILSTM-CCE and DeepeST-LSTM-CCE also showing excellent performance but with marginally lower scores and higher standard deviations.

Table 13 – Evaluation probabilities from Top-K Accuracy for the Foursquare New York dataset

Model	**ACC@1**		**ACC@5**		**ACC@10**	
	mean	**std**	**mean**	**std**	**mean**	**std**
DeepeST-BILSTM-CBCE	0,9987	0,0017	1,0000	0,0000	1,0000	0,0000
DeepeST-BILSTM-CCE	0,9970	0,0029	0,9990	0,0016	0,9993	0,0014
DeepeST-LSTM-CBCE	0,9983	0,0024	1,0000	0,0000	1,0000	0,0000
DeepeST-LSTM-CCE	0,9967	0,0022	0,9993	0,0014	0,9997	0,0011

Source: Created by the author

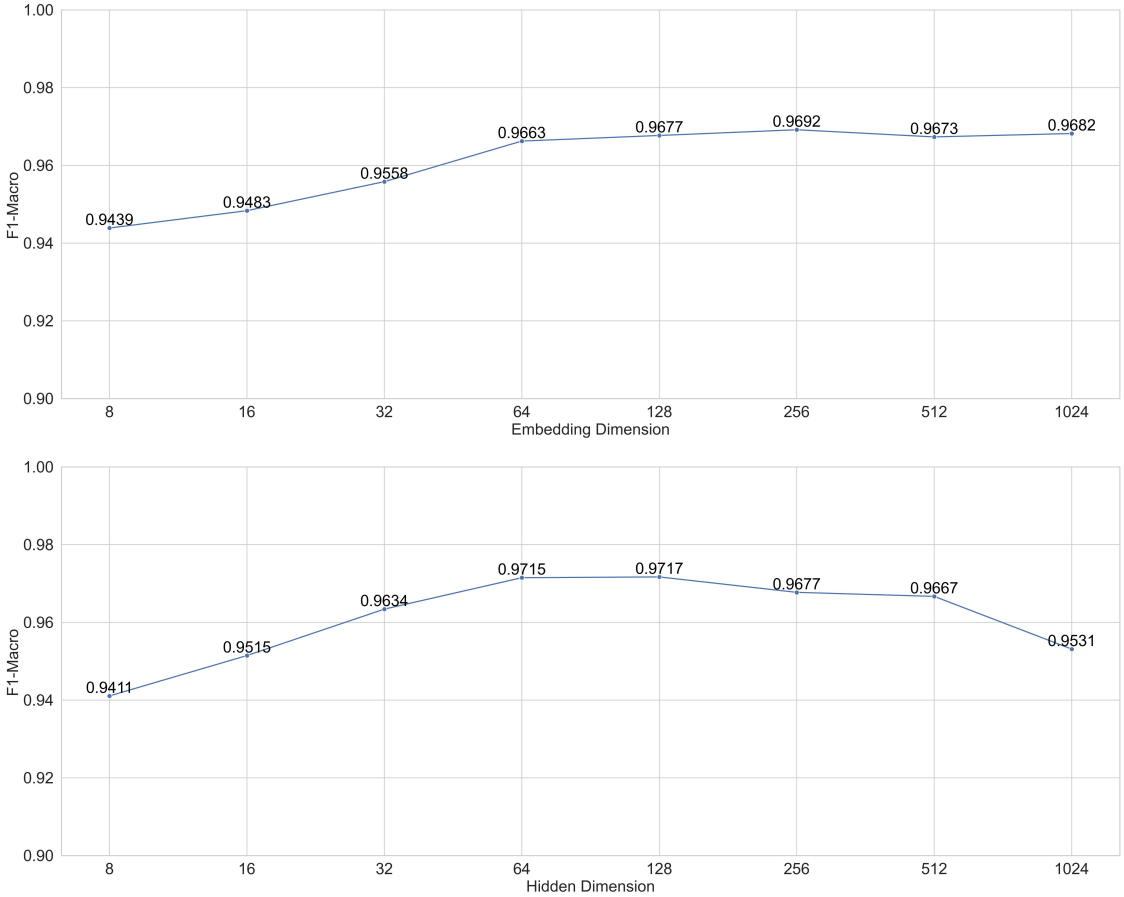
The DeepeST-BILSTM-CBCE variant consistently outperforms the other models across all datasets and evaluation metrics. The results suggest that using Class-Balanced Cross-Entropy (CBCE) with BILSTM layers provides a more practical approach for handling the challenges of trajectory user-linking in these multidimensional and imbalanced datasets. The low standard deviations across the board also highlight the robustness and stability of the DeepeST models.

4.3.5 Impact of Embedding Size and Hidden Dimension

We evaluated the performance of DeepeST through a parameter study to understand the effects of embedding dimension and hidden dimension, conducting experiments with the Gowalla dataset. We maintained a batch size of 64, a learning rate of 0.001, and a dropout rate of 0.5. Additionally, we set a cell size of 30m and defined the BILSTM and CBCE as the loss function.

The results are presented in Figure 14. Increasing the embedding and hidden dimensions improves the f1-score of DeepeST by allowing the model to store more information in the latent space. However, the performance curve stabilizes at specific values, and further increases in dimensions do not enhance the model's performance but add to the processing complexity.

Figure 14 – Classification results of varying embedding size and hidden dimension on DeepeST model for Gowalla.



Source: Created by the author

4.3.6 Impact of cell size for Grid Index

We conducted an additional study to test the impact of varying grid cell sizes on the performance of DeepeST with the Gowalla dataset. We maintained a batch size of 64, a learning rate 0.001, and a dropout rate 0.5. Additionally, we defined the BILSTM and CBCE as the loss function, and we tested thirteen different values to cell size, ranging from 10m to 500m, such as [10, 30, 60, 80, 110, 130, 150, 200, 250, 300, 350, 400, 500], as presented in Table 14. We highlighted the best value in **bold** and the second in *underlined*. The results have a few variations considering Balanced Accuracy, precision, recall, and f1-score. However, considering most metrics, the best value was a cell size of 30 meters for the Gowalla dataset.

Table 14 – Cell size variation of the Grid Index

Cellsize (meter)	Accuracy	Precision	Recall	F1-macro
10	0.9732	0.9790	0.9732	0.9726
30	**0.9784**	0.9815	**0.9784**	**0.9774**
60	0.9724	0.9760	0.9724	0.9702
80	0.9740	0.9794	0.9740	0.9725
100	0.9687	0.9759	0.9687	0.9687
150	0.9669	0.9755	0.9669	0.9669
200	0.9710	0.9803	0.9710	0.9711
250	0.9675	0.9742	0.9675	0.9677
300	0.9633	0.9725	0.9633	0.9624
350	0.9690	0.9769	0.9690	0.9688
400	0.9754	**0.9816**	0.9754	0.9747
450	0.9704	0.9784	0.9704	0.9687
500	0.9756	0.9810	0.9756	0.9750

Source: Created by the author

4.4 Experiment 03: Loss Functions in Deep Learning Model for TUL problem

In this section, we present a comparative study aimed at evaluating the performance of DeepeST using three different loss functions: Categorical Cross-Entropy (CCE), Class-Balanced Cross-Entropy (CBCE), and Focal Loss (FL). We assessed the performance and convergence speed of various DeepeST model variants across three datasets, each exhibiting different imbalance ratios (IR) and coefficients of variation (CV) derived from the Weeplace dataset.

We begin by providing details about the datasets, the DeepeST variations, and the evaluation metrics, followed by the experimental evaluation. Our case study aims to address the following objectives:

1. Assess the performance of DeepeST models using CCE, CBCE, and Focal Loss for the trajectory user-linking problem in handling class imbalance from check-in data.
2. Assess the impact of different imbalance ratios and coefficients of variation on model performance by creating three variations of the datasets.

4.4.1 Data Selection and Generation from Weeplace

We selected data from 2010, as most check-ins occurred more frequently in that year. From this data, we selected three databases with different imbalance ratios (IR) and Coefficient of Variation (CV) from the Weeplaces dataset. You can return to the subsection 3.3.5 for more details about IR and CV.

1. **Low IR of 2.0 and CV 0.17**: This dataset is a selection of 100 users based on the distribution of the highest data values from 2010. Specifically, we sorted users by the number of sub-trajectories and selected every 50th user (i.e., User 1, User 51, User 101, etc.), ensuring a consistent selection method while maintaining an imbalanced dataset.
2. **Medium IR of 4.9 and CV 0.36**: This dataset was created using a purely random sampling approach. We randomly selected 100 users from the 2010 database without considering their sub-trajectory counts or any other criteria. This method resulted in a dataset with a naturally occurring level of imbalance and variability, reflecting the inherent distribution of user activity.
3. **High IR of 10.6 and CV 0.37**: This dataset is a selection of 100 users based on the distribution of the highest values of data from 2010. We sorted users by the number of sub-trajectories and selected every 80th user (i.e., User 1, User 81, User 161, etc.). As observed, the greater the skipping interval in the selection process, the higher the Coefficient of Variation (CV), indicating increased variability and imbalance in the dataset.

4.4.2 Data Preparation

We use segmentation based on time and create weekly sub-trajectories from each user check-in, as performed in (FREITAS *et al.*, 2021c; May Petry *et al.*, 2020). We selected only sub-trajectories of users with at least 15 weekly sub-trajectories because we will have at least two samples for each user in the validation and test sets. For the Grid index approach in DeepeST, we created a virtual grid cell and set a cell size of 30m, covering all points for each dataset. Therefore, each latitude and longitude is mapped to a 30m x 30m region.

We apply the grid-search technique to combine several hyperparameters for each model to find the optimal set of hyperparameters. Although it is extremely computationally expensive and may take a long time to run your machine, grid-search ensures that we find the best hyperparameters considering each model and dataset. We keep 64 as the batch size, 0,001 as the learning rate for all the models, and 0,5 as dropout (**dp**). We also vary the units (**un**) of the recurrent layers and the embedding size to each attribute (**es**). We determine a first stopping callback, which is a stop training when, in our case, the accuracy has stopped improving. We set the early stopping as 30 for the patience argument to minimize overfitting, i.e., the number of epochs that produced the model's accuracy with no improvement, after which training should be

stopped. For further details, refer to Keras library [10].

To validate the models, we split the three datasets into training (80%), validation (10%), and test sets (10%). This split strategy ensures that the models have sufficient data to learn from (training set), tune hyperparameters to prevent overfitting (validation set), and evaluate the performance on unseen data (test set). Given that we are dealing with trajectories, we split the data using a temporal window for each user to preserve the sequence of events. We then ran the baseline algorithms ten times using the training and validation sets and evaluated the test set. The models were compared using balanced accuracy, precision macro, recall macro, and F1-Score macro. We specifically used Macro metrics because our dataset is imbalanced, and Macro averaging treats all classes equally, providing a more comprehensive evaluation of model performance across all classes. This approach ensures robust validation and reliable performance metrics.

4.4.3 Performance Comparison

There are six variations for DeepeST concerning the network layers and loss functions: three with BILSTM and three with LSTM. The variations with BILSTM are configured with different loss functions: one with Categorical Cross-Entropy (DeepeST-BILSTM-CCE), one with Class-Balanced Cross-Entropy (DeepeST-BILSTM-CBCE), and one with Focal Loss (DeepeST-BILSTM-FL). Similarly, the variations with LSTM are configured as follows: one with Categorical Cross-Entropy (DeepeST-LSTM-CCE), one with Class-Balanced Cross-Entropy (DeepeST-LSTM-CBCE), and one with Focal Loss (DeepeST-LSTM-FL).

From the results reported in Tables 15, 16, and 17, we summarize the performance comparison between the variants of DeepeST using different imbalance ratios (IR) and coefficient of variation (CV) for the Weeplaces dataset. We highlighted the two best values in **bold** and the third in *underlined*.

In Table 15, which evaluates the models with an IR of 2,0 and CV of 0,17, DeepeST-BILSTM-CBCE outperforms other models in terms of balanced accuracy (0,9114), precision macro (0,9273), recall macro (0,9114), and f1-Score macro (0,9108). DeepeST-LSTM-CBCE also shows strong performance with competitive values and achieves the best epoch time, indicating efficient convergence. DeepeST-BILSTM-CCE and DeepeST-LSTM-CCE also perform well, with slightly lower but still competitive values.

[10] https://keras.io/

Table 15 – Classification results on stratified holdout evaluation in the Dataset Weeplaces with IR 2,0 and CV 0,17

Model	Bal. Accuracy		Precision		Recall		F1-Score		Best Epoch	
	mean	std	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,9114**	0,0029	**0,9273**	0,0030	**0,9114**	0,0029	**0,9108**	0,0030	**8,70**	1,64
DeepeST-BILSTM-CCE	0,9086	0,0055	0,9244	0,0052	0,9086	0,0055	0,9074	0,0066	13,10	1,66
DeepeST-BILSTM-FL	0,9055	0,0053	0,9227	0,0055	0,9055	0,0053	0,9037	0,0064	11,90	1,79
DeepeST-LSTM-CBCE	**0,9102**	0,0052	**0,9274**	0,0038	**0,9102**	0,0052	**0,9095**	0,0051	**8,30**	2,50
DeepeST-LSTM-CCE	0,9082	0,0039	0,9230	0,0051	0,9082	0,0039	0,9067	0,0048	11,40	1,90
DeepeST-LSTM-FL	0,9008	0,0041	0,9187	0,0043	0,9008	0,0041	0,8995	0,0045	9,50	1,18

For the dataset with IR 4,9 and CV 0,36 (Table 16), DeepeST-BILSTM-CBCE maintains its superior performance with the highest balanced accuracy (0,7874) and recall macro (0,7874). DeepeST-BILSTM-CCE excels in Precision (0,8498) and F1-Score (0,7877). DeepeST-BILSTM-FL shows reasonable performance but lags slightly behind the top performers. DeepeST-LSTM-CBCE and DeepeST-LSTM-CCE also demonstrate strong performance across most metrics, with DeepeST-LSTM-CBCE achieving the best epoch time.

Table 16 – Classification results on stratified holdout evaluation in the Dataset Weeplaces with IR 4,9 and CV 0,36

Model	Bal, Accuracy		Precision		Recall		F1-Score		Best Epoch	
	mean	std	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,7874**	0,0066	0,8403	0,0112	**0,7874**	0,0066	**0,7830**	0,0077	12,40	2,12
DeepeST-BILSTM-CCE	0,7860	0,0072	**0,8498**	0,0118	0,7860	0,0072	**0,7877**	0,0059	21,20	2,86
DeepeST-BILSTM-FL	0,7784	0,0104	**0,8432**	0,0152	0,7784	0,0104	0,7731	0,0102	**12,00**	1,83
DeepeST-LSTM-CBCE	**0,7863**	0,0066	0,8373	0,0136	**0,7863**	0,0066	0,7802	0,0080	**9,40**	2,27
DeepeST-LSTM-CCE	0,7850	0,0069	0,8399	0,0095	0,7850	0,0069	0,7766	0,0066	18,70	3,59
DeepeST-LSTM-FL	0,7667	0,0120	0,8257	0,0116	0,7667	0,0120	0,7611	0,0097	13,40	2,41

In the case of IR 10,6 and CV 0,35 (Table 17), DeepeST-BILSTM-CBCE leads with the highest balanced accuracy (0,8115) and recall macro (0,8135). DeepeST-LSTM-CBCE demonstrates the highest precision macro (0,8591) and competitive F1-Score macro (0,7959). DeepeST-BILSTM-CCE and DeepeST-LSTM-FL also perform well across most metrics. DeepeST-BILSTM-CBCE and DeepeST-LSTM-FL have the best epoch times, highlighting their training efficiency.

Overall, the DeepeST models, especially those using BILSTM with CBCE and CCE loss functions, consistently perform well across three datasets and metrics. DeepeST-LSTM variants also show strong performance, particularly in precision and F1-Score. These results indicate that DeepeST models, especially those using BILSTM with CBCE and CCE loss functions, effectively handle the complexities of trajectory user-linking in imbalanced datasets. The use of Macro metrics ensures a fair and comprehensive evaluation, considering

Table 17 – Classification results on stratified holdout evaluation in the Dataset Weeplaces with IR 10,6 and CV 0,35

Model	Bal. Accuracy		Precision		Recall		F1-Score		Best Epoch	
	mean	std	mean	std	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,8115**	0,0109	0,8474	0,0105	**0,8135**	0,0108	0,7952	0,0146	12,10	2,38
DeepeST-BILSTM-CCE	0,8098	0,0070	**0,8613**	0,0119	0,8118	0,0070	**0,7974**	0,0069	15,10	1,91
DeepeST-BILSTM-FL	0,7907	0,0091	0,8414	0,0146	0,7930	0,0090	0,7780	0,0126	**11,00**	1,70
DeepeST-LSTM-CBCE	**0,8102**	0,0064	0,8494	0,0137	**0,8122**	0,0063	0,7946	0,0084	**11,80**	1,62
DeepeST-LSTM-CCE	0,8055	0,0088	**0,8591**	0,0093	0,8076	0,0088	**0,7959**	0,0081	12,50	2,22
DeepeST-LSTM-FL	0,7885	0,0139	0,8534	0,0177	0,7908	0,0138	0,7768	0,0168	12,60	2,07

the imbalanced nature of the data. DeepeST-LSTM-CBCE achieves the best epoch times across various datasets, reflecting its efficient convergence. This optimal epoch determination ensures the model generalizes well to new, unseen data, maintaining high accuracy and robustness. Focal loss also had rapid convergence compared to CCE in all datasets.

DeepeST implements CBCE, which provides several main advantages over Focal Loss in imbalance datasets:

- **Efficient Class Balancing:** CBCE effectively balances the classes by finding an adequate number of samples for each class using the data distribution from the training set. This ensures that the model does not become biased towards the majority class.
- **Simplified Training Process:** In Focal Loss, the training process can be more complex and time-consuming as the user needs to adjust the two parameters, alpha and gamma, using the validation set. This parameter tuning is not required for CBCE, making the training process more straightforward and faster.
- **Superior Performance:** CBCE consistently outperforms Focal Loss regarding balanced accuracy, recall, and f1-Score across the three datasets. This indicates that CBCE is more effective in handling the complexities of trajectory user-linking in imbalanced datasets.

4.4.4 Performance Evaluation using Top-K Accuracy

With such many classes, achieving exact predictions becomes increasingly complex it crucial to consider a broader set of potential outcomes. Analyzing ACC@1, ACC@5, and ACC@10 provides a deeper understanding of a model's ranking capabilities, offering insights into its practical effectiveness in scenarios where the correct class needs to be within the top K predictions, thereby complementing traditional metrics like precision, recall, balanced accuracy, and F1-score.

The K metric is calculated by sorting the predicted probabilities for each class in

descending order and checking if the true class label is within the top K predictions. For example, ACC@1 checks if the true class is the highest probability prediction, ACC@5 checks if the true class is among the top 5 predictions, and ACC@10 checks if the true class is among the top 10 predictions. This approach provides a more nuanced evaluation of the model's performance in ranking and identifying the correct class among its highest confidence predictions.

From the results reported in Tables 18, 19, and 20, we can summarize the performance comparison between the DeepeST model variants using different imbalance ratios (IR) and cross-validation (CV) splits for the Weeplaces dataset. We highlighted the two best values in **bold** and the third one in *underlined*.

Table 18 – Evaluation probabilities from Top-K Accuracy for the Weeplaces Dataset with IR 2,0 and CV 0,17

Model	ACC@1		ACC@5		ACC@10	
	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,9165**	0,0028	<u>0,9646</u>	0,0026	<u>0,9734</u>	0,0015
DeepeST-BILSTM-CCE	<u>0,9130</u>	0,0053	0,9644	0,0037	**0,9738**	0,0026
DeepeST-BILSTM-FL	0,9104	0,0050	0,9608	0,0031	0,9696	0,0025
DeepeST-LSTM-CBCE	**0,9153**	0,0049	**0,9651**	0,0036	0,9729	0,0027
DeepeST-LSTM-CCE	0,9123	0,0038	**0,9651**	0,0015	**0,9746**	0,0030
DeepeST-LSTM-FL	0,9047	0,0036	0,9538	0,0028	0,9676	0,0030

For the dataset with IR 2,0 and CV 0,17 (Table 18), DeepeST-BILSTM-CBCE demonstrates the highest Top-1 accuracy (0,9165) and strong performance in Top-5 (0,9646) and Top-10 accuracy (0,9734). DeepeST-LSTM-CBCE and DeepeST-LSTM-CCE also perform exceptionally well, particularly in Top-5 and Top-10 accuracy, showcasing the robustness of the CBCE and CCE loss functions in handling class imbalances.

Table 19 – Evaluation probabilities from Top-K Accuracy for the Weeplaces Dataset with IR 4,9 and CV 0,36

Model	ACC@1		ACC@5		ACC@10	
	mean	std	mean	std	mean	std
DeepeST-BILSTM-CBCE	**0,8102**	0,0066	**0,8917**	0,0055	**0,9271**	0,0083
DeepeST-BILSTM-CCE	0,8019	0,0059	**0,8953**	0,0066	**0,9230**	0,0086
DeepeST-BILSTM-FL	0,7983	0,0107	0,8914	0,0089	0,9213	0,0085
DeepeST-LSTM-CBCE	0,8022	0,0059	0,8909	0,0051	0,9219	0,0058
DeepeST-LSTM-CCE	**0,8083**	0,0068	0,8828	0,0088	0,9141	0,0099
DeepeST-LSTM-FL	0,7861	0,0122	0,8781	0,0082	0,9158	0,0112

For the dataset with IR 4,9 and CV 0,36 (Table 19), DeepeST-BILSTM-CCE leads in

Top-1 accuracy (0,8102) and Top-5 accuracy (0,8953). DeepeST-BILSTM-CBCE performs best in Top-10 accuracy (0.9271) and shows strong results in Top-1 and Top-5 accuracy. DeepeST-LSTM-CBCE and DeepeST-LSTM-CCE maintain competitive performance across all metrics, confirming the effectiveness of these loss functions.

Table 20 – Evaluation probabilities from Top-K Accuracy for the Weeplaces Dataset with IR 10,6 and CV 0,35

Model	**ACC@1**		**ACC@5**		**ACC@10**	
	mean	**std**	**mean**	**std**	**mean**	**std**
DeepeST-BILSTM-CBCE	0,8175	0,0083	0,8968	0,0089	0,9259	0,0097
DeepeST-BILSTM-CCE	**0,8222**	0,0079	**0,9003**	0,0091	**0,9294**	0,0089
DeepeST-BILSTM-FL	0,8015	0,0099	0,8892	0,0090	0,9222	0,0111
DeepeST-LSTM-CBCE	0,8190	0,0079	**0,9009**	0,0043	**0,9292**	0,0073
DeepeST-LSTM-CCE	**0,8198**	0,0084	0,8983	0,0113	0,9289	0,0112
DeepeST-LSTM-FL	0,8006	0,0120	0,8875	0,0112	0,9128	0,0067

For the dataset with IR 10,6 and CV 0,35 (Table 20), DeepeST-BILSTM-CCE shows the highest performance in Top-1 accuracy (0,8222), Top-5 accuracy (0,9003), and Top-10 accuracy (0,9294), DeepeST-LSTM-CBCE and DeepeST-LSTM-CCE also achieve excellent results, with DeepeST-LSTM-CBCE performing exceptionally well in Top-5 accuracy (0,9009) and Top-10 accuracy (0,9292). DeepeST-BILSTM-CBCE maintains strong performance across all metrics, while DeepeST-BILSTM-FL and DeepeST-LSTM-FL lag slightly behind.

Overall, DeepeST models utilizing CBCE and CCE loss functions consistently outperform those using Focal Loss across all Top-K accuracy metrics and imbalance ratios. This indicates that CBCE and CCE are more effective in handling class imbalances and providing robust performance in multi-class classification tasks.

4.4.5 Verification of Model Classification Using the Chi-Square Test

In this section, we present a statistical approach to verify whether our model's classification is significantly different from random classifications. We will use the Chi-Square Test to compare the confusion matrix of our model with samples of randomly generated confusion matrices (TALLARIDA *et al.*, 1987). The objective is to determine whether the distribution of correct and incorrect classifications of our model is consistent with that of a randomly classifying model or if there is statistical evidence that our model performs better.

The methodology adopted consists of the following steps:

1. **Obtaining the Model's Confusion Matrix:** First, we obtain the confusion matrix from applying our model to the test set. This matrix represents the model's actual distribution of correct and incorrect classifications. Since we have 100 users, the confusion matrix is a 100 by 100.
2. **Generating Random Confusion Matrices:** Next, we generate random samples of the confusion matrices. Each matrix is generated with the same number of instances as the confusion matrix of our model, ensuring comparability.
3. **Calculating Correct and Incorrect Classifications:** For each confusion matrix (both the model's and the sample's pseudo-randomly generated ones), we calculate the number of correct classifications (sum of the main diagonal) and the number of incorrect classifications (sum of the off-diagonal elements).
4. **Creating the Contingency Table:** We create a contingency table that combines the frequencies of correct and incorrect classifications for the model's matrix and the randomly generated matrices.
5. **Applying the Chi-Square Test:** We use the Chi-Square Test to determine whether there is a significant difference between the confusion matrix of our model and the random confusion matrices. This test allows us to assess whether our model's distribution of correct and incorrect classifications is statistically distinct from the distribution expected under random classification.

The output of the Chi-Square Test provides several key parameters:

- **Chi-Square Value (χ^2):** This value represents the test statistic. It quantifies the difference between the observed frequencies (in our confusion matrix) and the expected frequencies (assuming no significant difference). A higher χ^2 value indicates a larger discrepancy between the observed and expected frequencies.
- **p-value:** This value indicates the probability of obtaining a Chi-Square value as extreme as, or more extreme than, the one observed, assuming that the null hypothesis is true. The null hypothesis states no significant difference exists between the observed and expected frequencies. A low p-value (typically less than 0.05) suggests that the observed differences are unlikely to have occurred by chance, leading to the rejection of the null hypothesis (FIELD *et al.*, 2012).
- **Degrees of Freedom (DOF):** The degrees of freedom was 9801 for all tests. The degrees of freedom in the chi-square test are calculated based on the number of categories minus

one. Specifically, $(\text{number of rows} - 1) \times (\text{number of columns} - 1)$. For a 100-class classification problem, this calculation yields $(100-1) \times (100-1) = 99 \times 99 = 9801$ degrees of freedom. This accurately reflects the complexity and the scale of the multiple comparisons made in our analysis.

– **Expected Frequencies:** These values represent the frequencies that would be expected if there were no significant differences between the groups being compared. They are calculated based on the marginal totals of the contingency table and provide a baseline for comparison against the observed frequencies.

In Table 21, we present the chi-square (χ^2) test results, including the χ^2 values and p-values for three different datasets. The interpretation of these results is discussed below. Note that the χ^2 values presented are very high for all three datasets analyzed, indicating a substantial discrepancy between the observed and expected frequencies. The χ^2 value quantifies this difference: the higher the χ^2 value, the more significant the difference between the observed and expected distributions. In all cases, the χ^2 values exceed 79,000, which is significantly high. The p-value associated with each test is 0,00, less than the commonly used significance level of 0,05 (FIELD *et al.*, 2012). This means that the probability of observing such an extreme difference, assuming the null hypothesis is true, is extremely low. In other words, we reject the null hypothesis that no significant difference exists between the observed and expected distributions. Therefore, we conclude that there is a substantial difference between the model's classification and random classification for all datasets.

In summary, the chi-square test results for the three datasets indicate that the model's classifications differ significantly from random classifications. The high χ^2 values and p-values of 0,00 support the conclusion that our model classifies substantially better than random chance. This analysis provides robust statistical validation of the effectiveness of our classification model.

Table 21 – Chi-Square Test Results

Dataset	χ^2	**p-value**
Weeplaces with IR 2,5 and CV 0,17	79309,05	0,00
Weeplaces with IR 4,9 and CV 0,36	79080,71	0,00
Weeplaces with IR 10,6 and CV 0,35	79660,34	0,00

5 FINAL CONSIDERATIONS

This work investigates the trajectory classification problem and delves into the Trajectory User-Linking (TUL) challenge in location-based social networks (LBSN) in multidimensional and imbalanced datasets, aiming to categorize anonymous subtrajectories with specific users. We introduced a novel deep learning model named DeepeST (Deep Learning for Sub-Trajectory Classification), which employs embedding vectors inspired by natural language processing techniques to effectively manage data volume and sparsity designed to discern specific categories or users among various subtrajectories. DeepeST represents the inaugural model for trajectory classification tailored to address the challenges posed by imbalanced datasets adeptly.

We conducted three case studies to evaluate our DeepeST's performance against leading-edge methodologies in Machine Learning (Random Forest and XGBoost) for criminal activities and trajectory-user linking problem, Deep Learning (MARC, BITULER, and TUL-VAE) using LSBN datasets for trajectory-user linking problem, and DeepeST loss variations to imbalance datasets using LSBN datasets for trajectory-user linking problem, respectively. DeepeST achieved markedly higher balanced accuracy, precision, recall, and F1-score values across all experiments. DeepeST's generalizability is robust enough to accommodate a variety of trajectory classification issues within imbalanced domains, including transportation mode inference and next-stop prediction.

As for future directions, our initial focus will be on examining alternative loss functions across diverse scenarios. Specifically, we aim to develop a method for providing weights to the loss function to handle class imbalances. We also intend to augment DeepeST with additional functionalities, transforming it into a comprehensive framework for trajectory classification. While we achieved strong results across all datasets, we recognize the potential for further improvement, particularly with the Weeplaces dataset. To enhance accuracy, we plan to explore integrating additional deep learning techniques, such as attention mechanisms. Our approach is a supervised model to link users to anonymous trajectories. A limitation of supervised learning arises when new users, who were not represented in the training data, start using the application. Since the model has not been trained on data from these new users, it may struggle to predict their behaviors and preferences accurately. We must train a model periodically to add new user data to overcome this limitation.

Moreover, It would also be valuable to extract user patterns and classify them into profiles based on the features of their trajectories using unsupervised learning. These profiles

are essential for applications where new users are added to the classification problem. By developing robust profiling techniques, we can enhance the adaptability and scalability of our classification model, making it more effective in real-world scenarios where user behaviors and trajectories can vary widely. This reduces the need for extensive individual analysis, saving time and computational resources. Additionally, creating detailed user profiles based on trajectory features could improve personalized recommendations and targeted services, further increasing the practical utility of our research in various domains such as marketing, urban planning, and social network analysis.

REFERENCES

ABDI, H. Coefficient of variation. **Encyclopedia of research design**, v. 1, n. 5, p. 169–171, 2010.

BENNETT, K.; DEMIRIZ, A. Semi-supervised support vector machines. **Advances in Neural Information processing systems**, v. 11, 1998.

BERNDT, D. J.; CLIFFORD, J. Using dynamic time warping to find patterns in time series. In: **Proceedings of the 3rd international conference on knowledge discovery and data mining**. [*S. l.*: *s. n.*], 1994. p. 359–370.

BIAN, J.; TIAN, D.; TANG, Y.; TAO, D. Trajectory Data Classification. **ACM Transactions on Intelligent Systems and Technology**, v. 10, n. 4, p. 1–34, aug 2019. ISSN 21576904. Disponível em: https://doi.org/10.1145/3330138http://dl.acm.org/citation.cfm?doid=3344873.3330138.

BOGORNY, V.; RENSO, C.; AQUINO, A. R. de; SIQUEIRA, F. de L.; ALVARES, L. O. Constant-a conceptual data model for semantic trajectories of moving objects. **Transactions in GIS**, v. 18, p. 66–88, 2014. ISSN 14679671.

BOLBOL, A.; CHENG, T.; TSAPAKIS, I.; HAWORTH, J. Inferring hybrid transportation modes from sparse GPS data using a moving window SVM classification. **Computers, Environment and Urban Systems**, Elsevier Ltd, v. 36, n. 6, p. 526–537, 2012. ISSN 01989715. Disponível em: http://dx.doi.org/10.1016/j.compenvurbsys.2012.06.001.

BREFELD, U.; LASEK, J.; MAIR, S. Probabilistic movement models and zones of control. **Machine Learning**, Springer New York LLC, v. 108, p. 127–147, 1 2019. ISSN 15730565.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, 2001.

CANTURK, D.; KARAGOZ, P.; KIM, S.-W.; TOROSLU, I. H. Trust-aware location recommendation in location-based social networks: A graph-based approach. **Expert Systems with Applications**, Elsevier, v. 213, p. 119048, 2023.

CHANG, W.; SUN, D.; DU, Q. Intelligent sensors for poi recommendation model using deep learning in location-based social network big data. **Sensors**, MDPI, v. 23, n. 2, p. 850, 2023.

CHAO, P.; XU, Y.; HUA, W.; ZHOU, X. A survey on map-matching algorithms. In: SPRINGER. **Databases Theory and Applications: 31st Australasian Database Conference, ADC 2020, Melbourne, VIC, Australia, February 3–7, 2020, Proceedings 31**. [*S. l.*], 2020. p. 121–133.

CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. Smote: synthetic minority over-sampling technique. **Journal of artificial intelligence research**, v. 16, p. 321–357, 2002.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: . [*S. n.*], 2016. v. 13-17-Augu, p. 785–794. ISBN 9781450342322. Disponível em: http://dx.doi.org/10.1145/2939672.2939785.

CHEN, W.; HUANG, C.; YU, Y.; JIANG, Y.; DONG, J. Trajectory-user linking via hierarchical spatio-temporal attention networks. **ACM Transactions on Knowledge Discovery from Data**, Association for Computing Machinery (ACM), v. 18, p. 1–22, 5 2024. ISSN 1556-4681.

CHEN, W.; LI, S.; HUANG, C.; YU, Y.; JIANG, Y.; DONG, J. Mutual distillation learning network for trajectory-user linking. 5 2022. Disponível em: http://arxiv.org/abs/2205.03773.

CHEN, Y.; WANG, X.; FAN, M.; HUANG, J.; YANG, S.; ZHU, W. Curriculum meta-learning for next poi recommendation. In: **Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining**. [*S. l.*: *s. n.*], 2021. p. 2692–2702.

CHOLLET, F. **Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek**. [*S. l.*]: MITP-Verlags GmbH & Co. KG, 2018.

CUI, Y.; JIA, M.; LIN, T.-Y.; SONG, Y.; BELONGIE, S. Class-Balanced Loss Based on Effective Number of Samples. jan 2019. Disponível em: http://arxiv.org/abs/1901.05555.

DAMIANI, M. L.; HACHEM, F. **Segmentation techniques for the summarization of individual mobility data**. [*S. l.*]: Wiley-Blackwell, 2017.

DENG, D. Dbscan clustering algorithm based on density. In: IEEE. **2020 7th international forum on electrical engineering and automation (IFEEA)**. [*S. l.*], 2020. p. 949–953.

FANG, S.-H.; LIAO, H.-H.; FEI, Y.-X.; CHEN, K.-H.; HUANG, J.-W.; LU, Y.-D.; TSAO, Y. Transportation modes classification using sensors on smartphones. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 16, n. 8, p. 1324, 2016.

FENG, J.; ZHANG, M.; WANG, H.; YANG, Z.; ZHANG, C.; LI, Y.; JIN, D. Dplink: User identity linkage via deep neural network from heterogeneous mobility data. In: **The world wide web conference**. [*S. l.*: *s. n.*], 2019. p. 459–469.

FENG, Z.; ZHU, Y. A survey on trajectory data mining: Techniques and applications. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 4, p. 2056–2067, 4 2016. ISSN 21693536. Disponível em: http://ieeexplore.ieee.org/document/7452339/.

FERNÁNDEZ, A.; GARCÍA, S.; JESUS, M. J. del; HERRERA, F. A study of the behaviour of linguistic fuzzy rule based classification systems in the framework of imbalanced data-sets. **Fuzzy Sets and Systems**, v. 159, n. 18, p. 2378–2398, sep 2008. ISSN 01650114.

FIELD, A.; MILES, J.; FIELD, Z. **Discovering Statistics Using R**. SAGE Publications, 2012. ISBN 9781446258460. Disponível em: https://books.google.com.br/books?id=wd2K2zC3swIC.

FREITAS, N. C. A. de; SILVA, T. L. C. D.; MACêDO, J. A. F. D.; JúNIOER, L. M. Using deep learning for trajectory classification in imbalanced dataset. **The International FLAIRS Conference Proceedings**, v. 34, 4 2021. ISSN 2334-0762. Disponível em: https://journals.flvc.org/FLAIRS/article/view/128368.

FREITAS, N. C. A. de; SILVA, T. L. C. da; MACEDO, J. A. F. de; VASCONCELOS, L. C. M. de; JUNIOR, F. C. F. N. Crime monitor: Monitoring criminals from trajectory data. In: . [*S. l.*]: Institute of Electrical and Electronics Engineers (IEEE), 2021. p. 225–228.

FREITAS, N. C. de; SILVA, T. L. C. da; MACÊDO, J. A. F. de; JUNIOR, L. M.; CORDEIRO, M. G. Using deep learning for trajectory classification. 2021.

GAO, Q.; ZHOU, F.; ZHANG, K.; TRAJCEVSKI, G.; LUO, X.; ZHANG, F. Identifying human mobility via trajectory embeddings. In: **IJCAI**. [*S. l.*: *s. n.*], 2017. v. 17, p. 1689–1695.

GUO, G.; WANG, H.; BELL, D.; BI, Y.; GREER, K. Knn model-based approach in classification. In: SPRINGER. **On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings**. [*S. l.*], 2003. p. 986–996.

HAIXIANG, G.; YIJING, L.; SHANG, J.; MINGYUN, G.; YUANYUE, H.; BING, G. Learning from class-imbalanced data: Review of methods and applications. **Expert Systems with Applications**, Elsevier Ltd, v. 73, p. 220–239, 2017. ISSN 09574174. Disponível em: http://dx.doi.org/10.1016/j.eswa.2016.12.035.

HOCHREITER, S.; SCHMIDHUBER, J. Long Short-Term Memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997. ISSN 08997667.

HUANG, Z.; QIAO, S.; HAN, N.; YUAN, C.-a.; SONG, X.; XIAO, Y. Survey on vehicle map matching techniques. **CAAI Transactions on Intelligence Technology**, Wiley Online Library, v. 6, n. 1, p. 55–71, 2021.

KHAN, N. U.; WAN, W.; YU, S. Spatiotemporal analysis of tourists and residents in shanghai based on location-based social network's data from weibo. **ISPRS International Journal of Geo-Information**, MDPI, v. 9, n. 2, p. 70, 2020.

KIM, J.-S.; KAVAK, H.; ROULY, C. O.; JIN, H.; CROOKS, A.; PFOSER, D.; WENK, C.; ZÜFLE, A. Location-based social simulation for prescriptive analytics of disease spread. **SIGSPATIAL Special**, ACM New York, NY, USA, v. 12, n. 1, p. 53–61, 2020.

KOTSIANTIS, S. B. Decision trees: a recent overview. **Artificial Intelligence Review**, Springer, v. 39, p. 261–283, 2013.

LEE, J.-G.; HAN, J.; LI, X.; GONZALEZ, H. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 1, n. 1, p. 1081–1094, 2008.

LEITE, C.; PETRY, L. M.; BOGORNY, V. A survey and comparison of trajectory classification methods. **2019 8th Brazilian Conference on Intelligent Systems (BRACIS) (submitted)**, p. 788–793, 2019.

LIN, T.-Y.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLÁR, P. Focal Loss for Dense Object Detection. aug 2017. Disponível em: http://arxiv.org/abs/1708.02002.

May Petry, L.; Leite Da Silva, C.; ESULI, A.; RENSO, C.; BOGORNY, V. MARC: a robust method for multiple-aspect trajectory classification via space, time, and semantic embeddings. **International Journal of Geographical Information Science**, Taylor and Francis Ltd., 2020. ISSN 13623087.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013. Disponível em: http://ronan.collobert.com/senna/.

PATEL, D. Incorporating duration and region association information in trajectory classification. **Journal of Location Based Services**, v. 7, n. 4, p. 246–271, dec 2013. ISSN 17489725.

PATTERSON, D. J.; LIAO, L.; FOX, D.; KAUTZ, H. Inferring high-level behavior from low-level sensors. In: SPRINGER. **International Conference on Ubiquitous Computing**. [*S. l.*], 2003. p. 73–89.

RIZWAN, M.; WAN, W.; CERVANTES, O.; GWIAZDZINSKI, L. Using location-based social media data to observe check-in behavior and gender difference: Bringing weibo data into play. **ISPRS International Journal of Geo-Information**, MDPI, v. 7, n. 5, p. 196, 2018.

ROBUSTO, C. C. The cosine-haversine formula. **The American Mathematical Monthly**, JSTOR, v. 64, n. 1, p. 38–40, 1957.

SCHUSTER, M.; PALIWAL, K. K. Bidirectional recurrent neural networks. **IEEE Transactions on Signal Processing**, IEEE, v. 45, n. 11, p. 2673–2681, 1997.

SHARMA, L. K.; VYAS, O. P.; SCHIEDER, S.; AKASAPU, A. K. Nearest neighbour classification for trajectory data. In: SPRINGER. **Information and Communication Technologies: International Conference, ICT 2010, Kochi, Kerala, India, September 7-9, 2010. Proceedings**. [*S. l.*], 2010. p. 180–185.

SILVA, C. L. da; PETRY, L. M.; BOGORNY, V. A survey and comparison of trajectory classification methods. In: IEEE. **2019 8th Brazilian Conference on Intelligent Systems (BRACIS)**. [*S. l.*], 2019. p. 788–793.

TALLARIDA, R. J.; MURRAY, R. B.; TALLARIDA, R. J.; MURRAY, R. B. Chi-square test. **Manual of pharmacologic calculations: with computer programs**, Springer, p. 140–142, 1987.

TRAGOPOULOU, S.; VARLAMIS, I.; EIRINAKI, M. Classification of movement data concerning user's activity recognition via mobile phones. In: **Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14) - WIMS '14**. New York, New York, USA: ACM Press, 2014. p. 1–6. ISBN 9781450325387. Disponível em: http://dx.doi.org/10.1145/2611040.2611062.http://dl.acm.org/citation.cfm?doid=2611040.2611062.

VARLAMIS, I. Evolutionary data sampling for user movement classification. In: **2015 IEEE Congress on Evolutionary Computation (CEC)**. IEEE, 2015. p. 730–737. ISBN 978-1-4799-7492-4. Disponível em: http://ieeexplore.ieee.org/document/7256963/.

WANG, S.; BAO, Z.; CULPEPPER, J. S.; CONG, G. A survey on trajectory data management, analytics, and learning. **ACM Computing Surveys**, v. 54, p. 1–36, 3 2021. ISSN 0360-0300. Disponível em: https://dl.acm.org/doi/10.1145/3440207.

WANG, S.; CAO, J.; YU, P. S. Deep learning for spatio-temporal data mining: A survey. **IEEE Transactions on Knowledge and Data Engineering**, IEEE Computer Society, v. 34, p. 3681–3700, 8 2022. ISSN 15582191.

WANG, S.; LIU, W.; WU, J.; CAO, L.; MENG, Q.; KENNEDY, P. J. Training deep neural networks on imbalanced data sets. In: . IEEE, 2016. p. 4368–4374. ISBN 978-1-5090-0620-5. Disponível em: http://ieeexplore.ieee.org/document/7727770/.

WEI, H.; ZHANG, H. Research on hybrid recommendation algorithm for integrating consumption habits in lsbn. In: **Proceedings of the 2020 12th International Conference on Machine Learning and Computing**. [*S. l.*: *s. n.*], 2020. p. 188–192.

WEI, W. W. **Multivariate time series analysis and applications**. [*S. l.*]: John Wiley & Sons, 2018.

WEI, X.; QIAN, Y.; SUN, C.; SUN, J.; LIU, Y. A survey of location-based social networks: problems, methods, and future research directions. **GeoInformatica**, Springer, v. 26, n. 1, p. 159–199, 2022.

WINARNO, E.; HADIKURNIAWATI, W.; ROSSO, R. N. Location based service for presence system using haversine method. In: IEEE. **2017 international conference on innovative and creative information technology (ICITech)**. [*S. l.*], 2017. p. 1–4.

YAN, Y.; CHEN, M.; SHYU, M. L.; CHEN, S. C. Deep learning for imbalanced multimedia data classification. In: . [*S. l.*]: Institute of Electrical and Electronics Engineers Inc., 2016. p. 483–488. ISBN 9781509003792.

YANG, D. A deep learning-based fast route planning model for location-based social networks. **Journal of Circuits, Systems and Computers**, World Scientific, v. 32, n. 02, p. 2350028, 2023.

YANG, D.; ZHANG, D.; ZHENG, V. W.; YU, Z. Modeling user activity preference by leveraging user spatial-temporal characteristics in lbsns. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, Institute of Electrical and Electronics Engineers Inc., v. 45, p. 129–142, 2015. ISSN 21682232.

ZHAO, X.; TANG, J. Modeling temporal-spatial correlations for crime prediction. In: ACM. **Proceedings of the 2017 ACM on Conference on Information and Knowledge Management**. [*S. l.*], 2017. p. 497–506.

ZHENG, Y. Trajectory data mining. **ACM Transactions on Intelligent Systems and Technology**, v. 6, p. 1–41, 5 2015. ISSN 21576904.

ZHENG, Y.; LIU, L.; WANG, L.; XIE, X. Learning transportation mode from raw gps data for geographic applications on the web. In: ACM. **Proceedings of the 17th international conference on World Wide Web**. [*S. l.*], 2008. p. 247–256.

ZHOU, F.; GAO, Q.; TRAJCEVSKI, G.; ZHANG, K.; ZHONG, T.; ZHANG, F. Trajectory-user linking via variational autoencoder. In: **IJCAI**. [*S. l.*: *s. n.*], 2018. p. 3212–3218.

ZHOU, F.; YIN, R.; TRAJCEVSKI, G.; ZHANG, K.; WU, J.; KHOKHAR, A. Improving human mobility identification with trajectory augmentation. **GeoInformatica**, Springer, p. 1–31, 2019.